FINAL EVALUATION REPORT
Trusted Information Systems, Inc.
Trusted XENIX version 3.0

NATIONAL

COMPUTER SECURITY CENTER

9800 Savage Road
Fort George G. Meade
Maryland 20755-6000

8 April 1992

Report No. 92/001
C-Evaluation Report No. 23-92
Library No. S239,934

FOREWORD

This publication, the **Final Evaluation Report TIS Trusted XENIX version 3.0** is being issued by the National Computer Security Center under the authority of and in accordance with DoD Directive 5215.1, "Computer Security Evaluation Center." The purpose of this report is to document the results of the formal evaluation of TIS Trusted XENIX version 3.0 operating system. The requirements stated in this report are taken from **Department of Defense Trusted Computer System Evaluation Criteria**, dated December 1985.

Approved:

Patrick R. Gallagher, Jr.
Director,
National Computer Security Center

8 April 1992

This page intentionally left blank

TEAM MEMBERS

Team Members

Team members included the following individuals:

James Arnold	National Security Agency Fort George G. Meade, MD
William Havener Jasbir Singh	The MITRE Corporation McLean, VA

ACKNOWLEDGMENTS

The evaluation team acknowledges the great deal of effort put forth by previous evaluation team members, including the production of the bulk of this report:

Kenneth B. Elliott, III Dr. Dixie Baker	The Aerospace Corporation El Segundo, CA
Frank Belvin Sharon Kass	The MITRE Corporation Bedford, MA
Anthony Apted	National Security Agency Fort George G. Meade, MD

This page intentionally left blank

TABLE OF CONTENTS

EXECUTIVE SUMMARY

The security protection provided by Trusted Information Systems, Inc.'s Trusted XENIX version 3.0 operating system, configured according to the most secure manner described in the *Trusted XENIX System Administration Manual* [SYAD] and *Starting Trusted XENIX* [STXE], running on the hardware base described in Appendix A on page 103, has been examined by the National Security Agency (NSA), Trusted Product and Network Security Evaluation Division. The security features of Trusted XENIX were examined against the requirements specified by the *DoD Trusted Computer System Evaluation Criteria* (TCSEC) dated December 1985 in order to establish a candidate rating.

The NSA, Trusted Product and Network Security Evaluation team has determined that the highest class at which Trusted XENIX satisfies all the specified requirements of the Criteria is class B2.

A system that has been rated as being a B division system provides a Trusted Computing Base (TCB) that preserves the integrity of sensitivity labels and uses them to enforce a set of mandatory access control rules. The system developer has provided the security policy model on which the TCB is based and furnished a specification of the TCB and evidence that the reference monitor concept has been implemented.

The Trusted XENIX system consists of the Trusted XENIX operating system, version 3.0, running on the International Business Machines (IBM) Personal Computer AT (IBM PC/AT), IBM Personal System/2 (PS/2)[1], AST 386/25, GRiDCASE 1537, NEC PowerMate 386/25 or BusinessMate 386/25, Unisys Personal Workstation *2* Series 800/20C, or Zenith Z-386/33 Workstation. It is a multi-user, multi-tasking system which can support up to six concurrent users. Trusted XENIX contains many functional and security enhancements while maintaining binary compatibility with programs developed under IBM Personal Computer XENIX versions 1.0 and 2.0. Programs written according to the System V Interface Definition [SVID] are source-code compatible provided they use only the functions supported by Trusted XENIX.

In addition to providing user-specified access controls (i.e., discretionary access controls) through protection bits and access control lists, Trusted XENIX also provides the additional controls required to properly separate sensitive information from unauthorized users (i.e., mandatory access controls). In addition to the B2 requirements, Trusted XENIX provides the features associated with a B3 level of Discretionary Access Control, Trusted Path, and Trusted Facility Management.

[1]Models 50, 60, 70, 70T, 70P, or 80.

This page intentionally left blank

Chapter 1
Introduction

In December 1987, the National Security Agency (NSA) began a formal product evaluation of Secure XENIX, a product of International Business Machines Corporation (IBM). In June 1989, ownership of the product was transferred to Trusted Information Systems, Inc. (TIS) and the product renamed to Trusted XENIX. Subsequently, Trusted XENIX, version 2.0, was formally evaluated and placed on the Evaluated Products List (EPL) in January of 1991.

In June 1991, the NSA conducted a preliminary technical assessment of an effort by TIS to bring new hardware platforms (i.e., IBM PC/AT clones) into the evaluated configuration. An evaluation team was then assigned to evaluate the new configuration. The objective of that evaluation was to rate the Trusted XENIX version 3.0 system against the *Trusted Computer System Evaluation Criteria* (TCSEC), and to place it on the EPL with a final rating.

This report documents the results of this evaluation, which applies to the system as available from TIS in March 1992.

Material for this report was gathered by the NSA Trusted XENIX evaluation team from evaluation evidence for Trusted XENIX version 2.0 and through documentation, interaction with system developers, testing, and examination of the system source code.

1.1 Evaluation Process Overview

The Department of Defense Computer Security Center was established in January 1981 to encourage the widespread availability of trusted computer systems for use by facilities processing classified or other sensitive information. In August 1985 the name of the organization was changed to the National Computer Security Center. In order to assist in assessing the degree of trust one could place in a given computer system, the DoD TCSEC was written. The TCSEC establishes specific requirements that a computer system must meet in order to achieve a predefined level of trustworthiness. The TCSEC levels are arranged hierarchically into four major divisions of protection, each with certain security-relevant characteristics. These divisions are in turn subdivided into classes. To determine the division and class at which all requirements are met by a system, the system must be evaluated against the TCSEC by an NSA, Trusted Product and Network Security evaluation team.

The NSA supports the creation of secure computer products in varying stages of development from initial design to those that are commercially available. Preliminary to an evaluation, products must go through the Proposal Review Phase. This phase includes an assessment of the vendor's capability to create a secure system and complete the evaluation process. To support this assessment, a Preliminary Technical Review (PTR) of the system is done by the NSA. This consists of a quick review of the current state of the system by a small, but expert, team and the creation of a short report on the state of the system. If a vendor passes the Proposal Review Phase they will enter a support phase preliminary to evaluation. This support phase has two steps, the Vendor Assistance Phase (VAP) and the Design Analysis Phase (DAP). During VAP, the newly assigned team reviews design specifications and answers technical questions that the vendor may have about the ability of the design to meet the requirements. A product will stay in VAP until the vendor's design, design documentation, and other required evidence for the target TCSEC class are complete and the vendor is well into implementation. At that time, the support moves into DAP.

The primary thrust of DAP is an in-depth examination of a manufacturer's design for either a new trusted product or for security enhancements to an existing product. DAP is based on design documentation and information supplied by the industry source, it involves little "hands on" use of the system, but during this phase the vendor should virtually complete implementation of the product. DAP results in the production of an Initial Product Assessment Report (IPAR) by the NSA assessment team. The IPAR documents the team's understanding of the system based on the information presented by the vendor. Because the IPAR contains proprietary information and represents only a preliminary analysis by the NSA, distribution is restricted to the vendor and the NSA.

Products that have completed the support phase with the successful creation of the IPAR, enter formal evaluation. Products entering formal evaluation must be complete security systems. In addition, the release being evaluated must not undergo any additional development. The formal evaluation is an analysis of the hardware and software components of a system, all system documentation, and a mapping of the security features and assurances to the TCSEC. The analysis performed during the formal evaluation requires "hands on" testing (i.e., functional testing and, if applicable, penetration testing). The formal evaluation results in the production of a final report and an Evaluated Products List entry. The final report is a summary of the evaluation and includes the EPL rating which indicates the final class at which the product satisfies all TCSEC requirements in terms of both features and assurances. The final report and EPL entry are made public.

After completion of the Formal evaluation phase, products rated at B1 and below enter the rating maintenance phase (RAMP). The rating maintenance phase provides a mechanism to entend the previous rating to a new version of an evaluated computer system product. As enhancements are made to the computer product the ratings maintenance phase ensures that the level of trust is not degraded.

Rating Maintenance is accomplished by using qualified vendor personnel to manage the change process of the rated product during the maintenance cycle. These qualified vendor personnel must have strong technical knowledge of computer security and of their computer product. These trained personnel will oversee the vendor's computer product modification process. They will demonstrate to the Trusted Product and Network Security Evaluation Division that any modification or enhancements applied to the product preserve the security mechanisms and maintain the assurances required by the TCSEC for the rating previously awarded to the evaluated product.

1.2 Conventions

Throughout this report, filenames, command names, and system calls will be in *italics*. User and group names will be in *slanted* text. C language structures, their fields (including the term "u_area"), disk drive designators, and port designators will be in `typewriter` style. Intel 80286 or 80386 instructions and registers will be in `TYPEWRITER CAPITALS`. Signals and privileges will be in CAPITALS.

1.3 Document Organization

This report consists of four major sections, four appendices, and a bibliography. Section 1 is this introduction. Section 2 provides an overview of the system hardware and software architecture. Section 3 provides a mapping between the requirements specified in the TCSEC and the Trusted XENIX features and assurances that fulfill those requirements. Section 4 presents the evaluation team's comments on the system. The first two appendices identify specific hardware and software components to which the evaluation applies. The third appendix contains a list of acronyms used throughout the report, while the final appendix contains the Evaluated Products List entry for Trusted XENIX.

This page intentionally left blank

Chapter 2

System Overview

This section begins with a brief description of the hardware used in the system, followed by a short discussion of the history of the software. The remainder of the section describes in detail the security-relevant architecture and mechanisms used in Trusted XENIX.

2.1 Hardware Background

The evaluated system consists of the Trusted XENIX version 3.0 operating system running on an IBM Personal Computer/Advanced Technology (PC/AT), IBM Personal System/2 (PS/2), AST 386/25, GRiDCASE 1537, NEC PowerMate 386/25 or BusinessMate 386/25, Unisys Personal Workstation *2* Series 800/20C, or Zenith Z-386/33 Workstation[1]. These base systems are available in several configurations, each of which contains a fixed disk[2] as well as a diskette drive. Memory from 512 kilobytes (KBs) to 16 megabytes (MBs) in size may be used with the system. The user has several choices of CRT display, keyboards, as well as several printers. The Intel 80287 or 80387 Numeric Processor Extension is also available for use, if desired. As many as five ASCII terminals can be connected to a workstation, allowing a user to log in to Trusted XENIX from either the workstation keyboard, or from a connected terminal. Thus, as many as six separate users can make (interactive) use of the system at one time.

The Intel 80286 and 80386 central processing units (CPU) offer larger address spaces and improved performance with regard to the processors used in the earlier IBM PCs. More significantly, they incorporate features which support protected mode operation: hierarchical privilege levels (rings), privileged instructions, and memory segmentation and mapping, making it a usable hardware base for a trusted operating system.

2.2 Software Background

Trusted XENIX is a multi-user, multi-tasking operating system for microcomputers, based on the popular UNIX operating system.

UNIX was created by Ken Thompson and Dennis Ritchie at Bell Laboratories in 1969 to run on a PDP-7 computer. Since that time, UNIX has gone through many versions and updates, in the process becoming one of the most popular development environments in existence.

When PCs were first introduced, the most powerful operated with 8-bit CPUs, and were capable of supporting only simple operating systems. When 16-bit CPUs were introduced, more complex operating systems were possible, but such PCs would be heavily loaded when trying to run a full UNIX-like system. With the advent

[1]The AST 386/25, GRiDCASE 1537, NEC PowerMate and BusinessMate, Unisys Personal Workstation *2*, and Zenith Z-386/33 will be refered to collectively as "PC/AT Clones" in the remainder of this report.

[2]The GRiDCASE 1537 has a removable cartridge drive rather than a fixed disk.

of 32-bit CPUs, the power of PCs was able to accommodate a multi-tasking system such as UNIX. Since the name "UNIX" is licensed by AT&T, PC UNIX derivatives are named differently, but generally retain the "IX" suffix to identify the product as a UNIX derivative; hence "XENIX".

Microsoft Corp. introduced the first version of XENIX in August of 1980, based on version 7 of AT&T UNIX. When System III UNIX was introduced, Microsoft updated XENIX to incorporate the changes. The version on which Trusted XENIX is built is based on the Microsoft XENIX version which incorporates AT&T UNIX System V Release 2.0 features. The specific release of that XENIX was 2.0.

In June 1989, TIS acquired Secure XENIX from IBM, renamed it Trusted XENIX, and continued its development as a Class B2 system. Like its predecessor, Trusted XENIX is binary compatible with IBM PC XENIX for well-behaved applications and with most of the functions defined in the System V Interface Definition [SVID].

2.3 Operating Environment Background

Trusted XENIX is an IBM PC/AT, IBM PS/2, or PC/AT Clone product designed to provide a trusted workstation capability. When used as a trusted workstation, this system requires an operating discipline or work environment that is a hybrid of the traditional central site, time-sharing environment and the emerging personal workstation environment.

2.3.1 Physical Environment

Trusted XENIX is designed to operate in an independent, stand-alone mode, meaning one must view a trusted workstation as an independent element of an installation's entire computing facility. By moving the computing components into the user's work space, the TCB is moved into the user's work space. While this method of distributing the computing resources may be useful, it does not reduce the need to provide physical protection for the TCB. As the System Administration Manual [SYAD] points out, "some care must be exercised in the physical planning regarding workstation placement." The System Administration Manual goes on to state that the following rules must be followed when planning workstation placement:

> "Worthy of particular note is the prevention of physical modification to the system unit. Unlike traditional centralized computer systems where users are physically separated from the majority of system hardware, the operational environment typical for a workstation requires users to be physically co-located with most, if not all, system hardware components. Therefore, it is necessary for the site to provide adequate physical controls to ensure that the system hardware may only be modified by appropriate administrative personnel who are cleared for all data located on the system."

2.3.2 Personnel Requirements

Trusted XENIX depends on a number of "administrative users" to install, maintain, and operate the system properly. Five important roles must be fulfilled by these administrative users. These five roles are defined in the System Administration Manual as Trusted System Programmer (TSP), System Security Administrator

(SSA), Auditor, Secure Operator (SO), and Accounts Administrator (AA). The System Administration Manual expects that no one individual would fill all five roles, and it expects that the normal workstation user would not fill any of these roles.

- Trusted System Programmer (TSP)

  The TSP is responsible for initial system installation to include set-up and installation of the hardware, installation of the operating system, and initializing all of the site-dependent system databases (e.g., peripheral configuration, administrative users accounts, physical security environment). In addition, the TSP is responsible for most system maintenance activities and is the only person that is permitted to operate the system in maintenance mode. The TSP role does not exist in multi-user mode.

- System Security Administrator (SSA)

  The SSA is responsible for the security relevant activities required for day-to-day operations. These activities include updating configuration information (e.g., peripheral security characteristics), verifying consistency of important TCB databases (through use of the *scheck* command), maintaining user authentication data and limits as well as adding and deleting users, and adjusting system security parameters (e.g., maximum log in time). SSAs are distinguished from other users by being members of the *ssa* group.

- Auditor

  The Auditor is responsible for maintaining and reviewing the system audit log. The Auditor's duties include turning system auditing on and off, identifying which events are to be audited, adjusting system audit thresholds, ensuring that there is sufficient space for the audit log, and producing and reviewing audit reports. Auditors are distinguished from other users by being members of the *audit* group.

- Secure Operator (SO)

  The SO is responsible for the daily tasks which require more privilege than is allotted ordinary users. On mainframe systems these tasks would be performed by an independent operations staff. The SO's duties include managing shared devices and the queues, if any, associated with them, and file system backup and restore. SOs are distinguished from other users by being members of the *so* group. No other operator capability is required or provided.

- Accounts Administrator (AA)

  The AA is responsible for managing the system accounting function. The AA's duties include turning on and off accounting of various types of activities and producing the required accounting reports. AAs are distinguished from other users by being members of the *aa* group.

Trusted XENIX does not use the user id (uid) attribute of a process to distinguish it as acting on behalf of an administrative user. All special uids that are required by the system are not valid uids for use during log in. This restriction is enforced by setting the passwords for these uids to invalid values and preventing these passwords from being changed.

2.4 Trusted XENIX TCB Definition

The TIS Trusted XENIX Trusted Computing Base (TCB) consists of five main components:

Hardware	the Intel 80286 or 80386 CPU, memory, controllers, and peripherals of the IBM PC/AT, IBM PS/2, or PC/AT Clone system,
Firmware	the permanent code in the ROM BIOS, keyboard microcontroller, and peripheral controllers of the IBM PC/AT, IBM PS/2, or PC/AT Clone system,
Kernel software	the Trusted XENIX software that runs with privilege level zero (ring zero) of the Intel 80286 or 80386 CPU
Trusted Processes	the Trusted XENIX software that runs with privilege level three (ring three) as independent processes with software-defined privileges,
Administrative files/databases	the data structures which define users, security levels, auditing, and related information necessary to define the secure state of the Trusted XENIX system.

The following sections cover in detail both the hardware and software components of the TCB. Firmware is discussed in the hardware section, while administrative files and databases are covered in the software section.

2.5 Trusted XENIX Hardware

This section describes the important characteristics of all system hardware, firmware and diagnostics, and is broken down into seven subsections. First, the IBM PC/AT Architecture is presented, followed by the IBM PS/2 Architecture. The third and fourth subsections discuss the Intel 80286 and 80386 Microprocessors. Finally, Input/Output (I/O) Devices, System Board Firmware, and Physical Protection are discussed.

2.5.1 IBM PC/AT Hardware Configuration

Trusted XENIX operates on a standard IBM PC/AT, with no hardware modifications. The IBM PC/AT models included are: 5170 Model 099, 5170 Model 239, and 5170 Model 339.

All three models include a system unit, keyboard, diskette drive and a fixed drive. All three models also include the following items on the system ("mother") board:

- Intel 80286 operating at 8 MHz clock speed (the Model 099 operates at 6 MHz)
- 64 KB Read-only Memory (ROM)
- 512 KB of dynamic Random Access Memory (RAM)
- System Clock/Calendar
- Complementary Metal Oxide Semiconductor (CMOS) RAM containing system configuration switches
- Battery back-up for CMOS RAM and Clock/Calendar

- Socket for Intel 80287 Numeric Processor Extension
- Seven Direct Memory Access (DMA) channels
- Sixteen level interrupt handler
- Eight I/O expansion slots

The I/O expansion slots are designed to support devices (i.e., I/O boards) with either 8- or 16-bit data paths; the 8-bit data paths are included to support PC and PC/XT compatible boards. All disk and tape drive controllers are connected to I/O expansion slots supporting 16-bit data paths, while the serial/parallel device adapters are connected to slots supporting only an 8-bit data path. This bus architecture is now referred to as "Industry Standard Architecture," or ISA.

2.5.2 PC/AT Clone Hardware Configurations

Trusted XENIX also operates on a AST 386/25, GRiDCASE 1537, NEC PowerMate or BusinessMate, Unisys Personal Workstation *2*, or Zenith Z-286/33 (the PC/AT Clones), with no hardware modifications.

Each of the PC/AT Clones is based either on an Intel 80286 or 80386 operating at speeds between 16 and 33 Mhz. Each PC/AT Clone is built around a standard ISA bus, including a system board with components very similar to those found on an IBM PC/AT[3], at least one diskette drive, at least one fixed disk drive[4], keyboard, and monitor. Additionally, each of the PC/AT Clones (except the GRiDCASE 1537) includes a built-in memory cache. Each of these caches works on absolute addresses provided by the CPU, mapping multiple addresses to a single location in the cache. The implementation of each cache ensures that only valid cache data can be read and modified cache data is written to memory prior to being overwritten.

AST 386/25

The AST 386/25 is a desktop machine utilizing an Intel 80386 at 33Mhz.

GRiDCASE 1537

The GRiDCASE 1537 is a portable laptop machine, with a built-in keyboard and electroluminescent light-emitting flat-panel display, utilizing an Intel 80286 at 16Mhz.

NEC PowerMAte and BusinessMate

The PowerMate is a desktop workstation and the BusinessMate is a desk-side workstation, both machines utilize an Intel 80386 at 25Mhz.

[3]A more precise list of components can be found in "Evaluated Hardware Components", page 103.

[4]The GRiDCASE 1537 has a removable cartridge drive rather than a "fixed" disk drive. However, for consistency, the generic term "fixed disk drive" will include the removable cartridge drive.

Unisys Personal Workstation *2*

The Personal Workstation *2* is a desktop workstation utilizing an Intel 80386 at 20Mhz.

Zenith Z-386/33

The Zenith Z-386/33 is a desktop workstation utilizing an Intel 80386 at 33Mhz.

Direct Memory Access

The DMA controller (on all evaluated machines) allows I/O devices to transfer data directly to and from memory, freeing the system microprocessor of I/O tasks. The system microprocessor provides the memory address and I/O address for the transfer as well as a transfer count. In this manner, the system microprocessor can control the ability of a process to transfer data into and out of another process' domain (i.e., memory locations outside the process' address space).

Interrupt Controller

The IBM PC/AT and PC/AT Clones provide hardware support for the 80286 and 80386 microprocessors' non-maskable interrupt (NMI) and two Intel 8259A Controller chips provide 16 levels of system hardware interrupts.

The 16 interrupt levels may also be "shared" by multiple devices, so long as each interrupt handler adheres to the appropriate shared interrupt logic. When a handler gains control as a result of an interrupt, it determines whether its device was the device that actually caused the interrupt. If so, it handles the interrupt. If not, the handler passes control to the next interrupt handler in the chain.

I/O Channel

The I/O Channel supports the DMA and Interrupt capabilities, as well as external adapter cards. The I/O Channel is essentially a bus which permits the access of memory and I/O devices within the system. The I/O Channel includes address lines, data lines, system clock, interrupt, and other control signals.

2.5.3 IBM PS/2 Hardware Configuration

Trusted XENIX also operates on a standard IBM PS/2 with no hardware modifications. The IBM PS/2 models included are: 50, 60, 70, 70P, 70T, and 80.

All six models include a system unit, keyboard, diskette drive and a fixed drive. IBM PS/2 models 50 and 60 include the following items on the mother board:

- Intel 80286 operating at 10 MHz clock speed

- 128 KB ROM
- 1 MB of dynamic RAM, with parity, one bit per byte
- System Clock/Calendar
- CMOS RAM containing system configuration switches
- Battery back-up for CMOS RAM and Clock/Calendar
- Socket for Intel 80287 Numeric Processor Extension
- Eight DMA channels
- Parallel Port
- 16-bit channel
- EIA RS-232-C serial communications controller and port
- Sixteen level interrupt handler
- Four Micro Channel connectors
- Integrated video graphics subsystem

The IBM PS/2 model 60 includes an extra 2 KB RAM extension with battery backup and an extra four Micro Channel connectors (for a total of eight).

The IBM PS/2 model 70 includes all of the above with the following differences:

- The CPU is and Intel 80386 operating at 16, 20, or 25 Mhz (rather than an Intel 80286)
- More dynamic and CMOS RAM (up to 8 MB and 2 MB respectively) is available
- 32-bit channels are supported, in addition to the 16-bit channels
- There are eight, rather than four, Micro Channel connectors
- There is an 80387 Numeric Processor Extension socket

The IBM PS/2 model 70T is a TEMPESTed version of the model 70. The model 70P is a portable version of the model 70, with a plasma display. The 70P has a single available clock speed of 20 MHz.

The IBM PS/2 model 80 falls somewhere between the Model 50 and 60 PS/2s and the Model 70. The following list describes the differences between the Models 70 and 80:

- The CPU is and Intel 80386 operating at 16 and 20 Mhz
- The dynamic RAM space (up to 2 MB) is less than a Model 70 but more than the others

The DMA, Interrupt Controller, and I/O Channel features discussed previously are also applicable to the IBM PS/2 machines.

Micro Channel Architecture

The PS/2 Architecture is built upon IBM's Micro Channel architecture which consists of an address bus, data bus, transfer control bus, arbitration bus, and support signals. This is distinct from, and not compatible with, the ISA bus of the IBM PC/AT. Micro Channel uses synchronous and asynchronous communication to permit control and data transfers between memory, I/O devices, and the controlling master. The controlling master can be a DMA controller, system microprocessor, or a bus master.

The Micro Channel architecture allows for bus ownership to be controlled by a central arbitration point, which acknowledges up to 16 devices. The central arbiter gives devices the ability to share and control the system. It allows burst data transfers and prioritization of control between devices. The devices can be DMA slaves, bus masters, or the system microprocessor.

Arbitrating devices request use of the system channel. The central arbiter initiates an arbitration cycle when the present device releases the channel. The requesting devices then drive their assigned 4-bit arbitration level onto the arbitration bus. When a device sees a more significant request on the arbitration bus, it stops driving the bus. The device with the most significant request thereby wins control of the channel when the arbiter goes to the grant state. A device may transfer multiple times, unless another device requests use of the channel, in which case further transfers are postponed until the device wins the channel again.

This design permits the central arbiter to indicate when the channel is available for requests and to allow preemption when needed. The design also requires each device to be aware of and adhere to the logic used by the arbitration bus. The arbitration level used by a device can be assigned by diagnostic or system programs.

Programmable Option Select

The Programmable Option Select eliminates the need for switches on the system board and adapters by replacing their function with programmable registers. System configuration utilities create configuration data for the system board and each adapter. This is achieved by reading a unique adapter ID number from each adapter, matching it with an adapter description file, and configuring the system accordingly. The resulting data and adapter IDs are stored in battery-backed RAM.

These data permit the power-on self-test (POST) to configure the system whenever the system is powered on. The POST first verifies that the system has not changed by reading the adapter ID numbers and comparing them with the previously stored values. If the configuration has changed, the system configuration utilities must be rerun.

Configuration utility programs are provided with the IBM PS/2 on the Reference Diskette. Since these utilities allow the hardware-recognized configuration to be altered, the Trusted Systems Programmer (TSP) must restrict the use of this diskette to authorized users (see section 2.5.8).

2.5.4 Intel 80286 Characteristics

The 80286 has a 24-bit address path, and a 16-bit data path, both externally and internally. The 80286 operates in one of two addressing modes: Real Address Mode and Protected Virtual Address Mode. Trusted XENIX uses the Real Address Mode only during the initialization process; otherwise, it operates exclusively in the latter ("protected") mode until it is shut down. The description of the processor's operation which follows assumes operation in protected mode.

Security features which have been incorporated into the 80286 in protected mode include the provision of hierarchical privilege levels, the mediation of all memory accesses through a central mechanism, and other features oriented toward providing process separation. Intel incorporated security features into its design, and conducted design analyses to determine the performance of the planned security environment, as described in [BAUE].

Segment Access Control

All memory references, regardless of the processor privilege level, are made through a single, common mechanism. This mechanism provides an executing process access to up to four active memory segments, using segment selector registers:

- CS — Code segment selector
- DS — Data segment selector
- SS — Stack segment selector
- ES — Extra segment selector

A segment is from 1 byte to 64 KB in length, and is the fundamental unit of storage of the processor. An instruction may use a displacement or offset to point to a particular byte or word within the segment, or it may use an additional register to modify the effective address. In either case, the address is interpreted as being within the segment determined by the selector. All of a process' references to memory use these selectors, either explicitly or implicitly. The selectors thus provide a virtual address for the segment; its actual address is provided through the use of descriptors, which contain the starting address and length of the segment, as well as hardware access control information (the "access rights").[5]

The process' virtual memory is composed of segments, which are addressed through the segment selectors. In order to simplify the management of multiple processes which share some common operating system code, this virtual memory is divided into two classes: one for private code and data, and the other for code and data shared with other processes. The segment descriptors for these two classes are maintained in different tables: a Global Descriptor Table (GDT) for shared code and data, and a Local Descriptor Table (LDT) for private code and data. All processes share a single GDT, but each has its own LDT. These tables contain descriptors for all segments which the process is allowed access. The segment selector value (the virtual address) implicitly chooses which of these two tables will be used; the protection operation of the system is independent of whether a descriptor is in the LDT or the GDT.

All LDTs must be maintained as segments within the GDT. The GDT is implicitly also a segment in the GDT, but there is no descriptor for it; it is pointed to directly by a dedicated GDT Register (GDTR). Besides segment descriptors, the tables also contain special-purpose control descriptors which are used for procedure

[5]To avoid the performance penalty of multiple memory references per request, the selector registers are extended to include the segment's actual address, length, and access rights. This extension acts as a translation look-aside buffer, which the processor uses in all memory references with that selector. The extension information is obtained from main memory the first time a reference is made after a segment selector has been loaded with a new value. The extension is not directly addressable by processes, regardless of the process' privilege level.

calls, task switching, and interrupt handling. The four types of descriptors used are data, executable (code), system segment, and gate segment descriptors.

Trusted XENIX uses two of the four privilege levels defined for the 80286; they are numbered from zero to three in decreasing order of privilege, one and two being the unused privilege levels. Privilege levels are associated with all descriptors and selectors and thus are associated with segments, processes and gates. Selectors are 16 bits in length: 1 bit is used to select the GDT or LDT; 13 bits are used to name (or address) the virtual segment; and the remaining 2 bits contain the requested privilege level (RPL), whose interpretation depends on the selector type. When a process executes, the RPL[6] of its CS determines the current privilege level (CPL)[7] of the process. The RPL of the CS and the SS must be equal. When the process executes an instruction to load one of the segment selectors, checks are made using the CPL and the access rights field within the selected descriptor, to ensure that the operation is permitted by the protection rules. Initial checking ensures that the descriptor exists and is well-formed, and that the segment it points to is present. The remaining checks use the access rights field of the descriptor, the CPL, and are dependent on which segment selector is chosen.

The access rights field of a segment descriptor includes the descriptor privilege level (DPL), and control information to distinguish the following cases:

- data segment: read-only or read/write
- code (executable) segment: execute-only or execute/read

In addition, a code segment can be designated as a conforming code segment[8] in which case a different check is made. Trusted XENIX makes no use of conforming code segments. A data segment can be described in terms of its length measured down from its highest address (64 KB), instead of its maximum length measured up from its lowest address, which is useful for stack segments; this does not have any additional protection implications and will be ignored here. Access rights checks that are made are as expected; for instance, only the descriptors of code segments can be loaded into the CS, and only those of writable data segments into the SS, while descriptors for data segments, as well as those for any code segments with the read access right, can be loaded into the DS and ES. Privilege level checks are also straight-forward: for CS and SS, DPL must equal CPL; while for DS and ES, DPL $\geq$ CPL. This means that control can be transferred by `CALL` and `JMP` instructions only within the same privilege level, and only data at the same or a less privileged level (higher-valued DPL) can be read or written. After these checks have been made, the segment selector is loaded; on the next memory reference which uses the selector, the processor loads the associated cache (translation look-aside buffer) register with the descriptor, thereby speeding up future memory references.

Once the segment selector has been loaded, the protection checks to determine whether access is permitted have been accomplished. However, further checks are necessary when a process references a memory location during subsequent process execution. These checks are that the displacement generated by the execution of an instruction is within the length limit of the segment, and that no attempt is made to write into a read-only segment. Any violation causes an interrupt before any memory reference is started and before any registers are modified, leaving the process state intact for restarting the instruction, say, if the segment length is subsequently increased. The limitation that code which is marked execute-only cannot be accessed using DS or ES, extends to prefixing (the substitution of a segment selector for the normal one in an instruction), so such code cannot be read using the CS override prefix on a data movement instruction.

[6]The RPL of a code segment is not directly modifiable to the resulting process.

[7]In keeping with Intel's nomenclature, the term CPL is used throughout this section to mean "Current Privilege Level", rather than "Current Process Level", as used in the other parts of this report.

[8]Please refer to [286 85] or [386 86] for a more complete description.

An additional level of protection is provided for pointer validation using the RPL. This facility is provided for use by more privileged procedures which need to validate that the pointers passed to them by less privileged callers are confined to the privilege level of the caller. To use this facility, the procedure sets the RPL of its DS or ES selector to the privilege level of its caller (or whatever level it chooses.) The access checks then made for DS or ES selectors will use the greater of RPL and CPL in the comparison with the DPL; thus access can be made more restrictive, at the process' choosing, than would otherwise occur, so that the caller's attributes, rather than those of the more privileged procedure, are used in making access checks. Additional aids are provided to procedures in the form of instructions that set the RPL directly from a given selector value (for example, the caller's CS), and for verifying the access capabilities of a pointer prior to using it.

Hardware Address Space Structure

Figure 2.1 shows an overview of the hardware's internal view of memory. The 80286 and 80386 have a number of registers that point to memory resident structures used in defining a process' address space (e.g., TSS, IDT, GDT, LDT), as shown. Of these, the TSS and LDT are process specific, and the others are common to all processes. Whenever a context switch occurs, either the contents of the structure must be changed (i.e., TSS) or the pointer to the structure must be modified to point to the new structure (i.e., LDT). A process does not access the TSS, IDT, LDT, or GDT directly, but rather uses them indirectly for determining physical addresses of virtual segments, among other things (e.g., defining which privilege level a segment is protected by).

A process' address space is defined as being a number of segments (up to four at any given time) which the process can access. These segments are defined indirectly through the selector registers: CS (code segment selector), DS (data segment selector), SS (extra segment selector), and ES (extra segment selector). These selectors are treated specially by different instructions (e.g., all instructions are fetched via the CS, while all push and pop operations use the SS). Each of these selectors indicates whether the LDT or GDT is to be used, and includes an index into that table that will result in a descriptor for the desired segment. The descriptor includes a physical pointer to the segment, and the size of that segment. Hence, the entire possible address space of a process is defined by the segments defined by the descriptors in the GDT and the LDT, and the instantaneous address space of a process is defined by the descriptors referred to by the four selectors.

Control Transfers

Control transfers within a segment (so-called "short jumps"), and between segments at the same privilege level ("long jumps"), use the facilities of the segment selector described above. Control transfers between segments at different privilege levels are treated specially, as are control transfers involving task switching. Only calls and returns (and interrupts) are permitted across privilege level boundaries. To effect these, a special descriptor type, called a gate, is provided. Control transfer instructions call the gate rather than attempting to transfer directly to the code segment; such direct transfers of control are prevented by the processor. Gates are of four types: call, task, interrupt and trap gates. All four gates define a new address to which control is transferred when they are invoked; this address is not normally directly accessible to the calling program. That is, an attempt to load a gate descriptor directly into a segment selector generates a protection fault. Gate transfers use the mechanism of the task and the task state segment (TSS). Of the four types of gates available, Trusted XENIX uses a single call gate for all direct kernel calls and a number of interrupt and trap gates to service interrupts and traps.

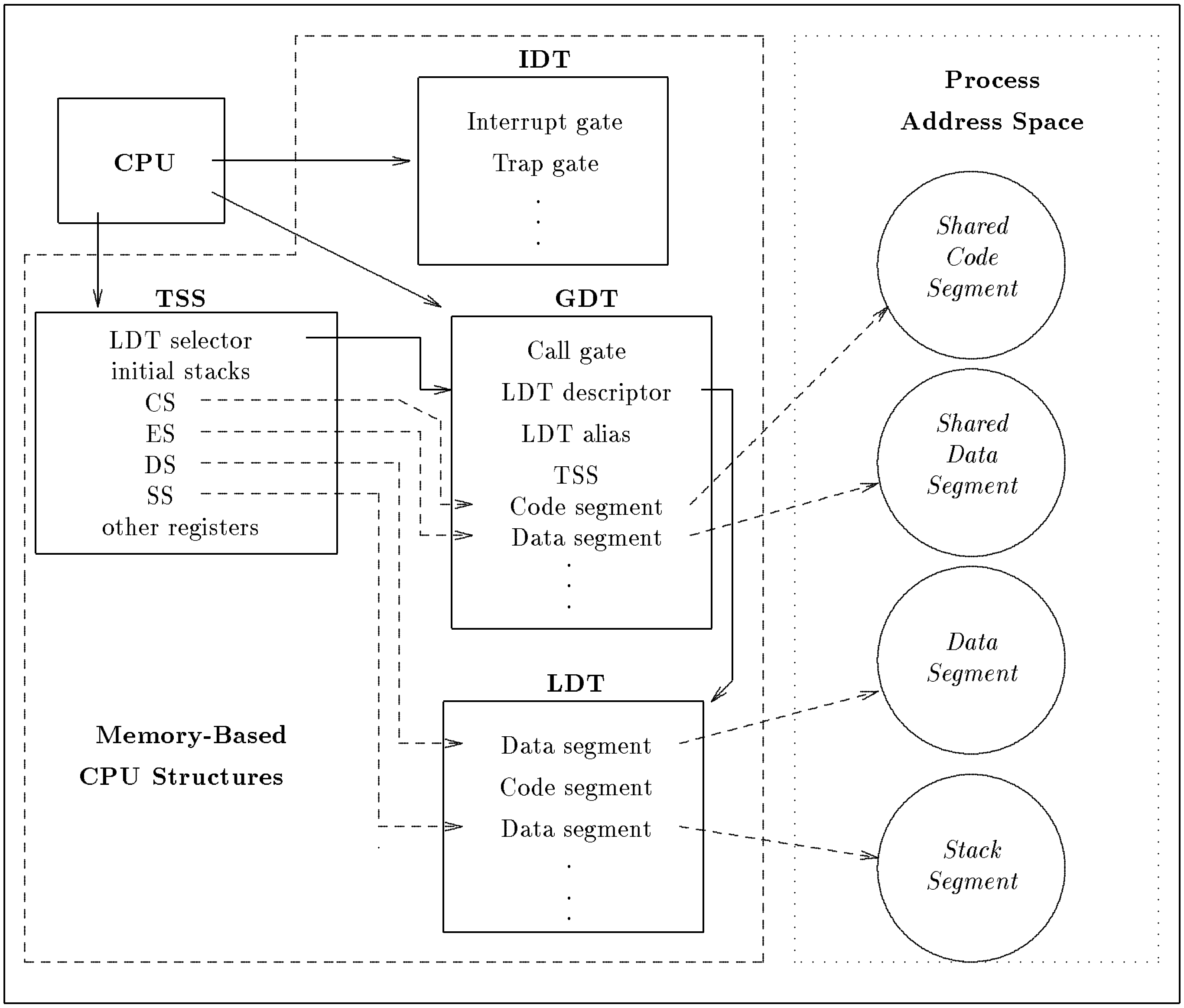

Figure 2.1. Hardware View of Memory Space

Tasks and Task Gates

The 80286 knows of a task as a single thread of execution. When a task is defined (initiated), a TSS is created which defines the save area for the task, and also defines the stacks to be used for task switching at privilege levels zero, one and two.[9] The current TSS is referenced in the special task register (TR). The TSS has a special descriptor containing its base address and limit, and its DPL. (A task gate is a descriptor which points to a TSS; it provides an additional level of indirection, and can be used, for example, to provide access selectively to TSSs which are otherwise hidden in the GDT.) Access to a TSS is controlled so that its static portions (those containing the stack pointers for each privilege level, and the LDT) never change during task switching; its dynamic portion contains space for all the program-visible registers and flags, and task switching automatically saves and restores all these registers. There is only a single TSS, and it is not used after system start-up.

Call Gates

When a call gate is used to make an inward transfer of control (moving to a more privileged level), access to the gate is checked using the same rules of privilege as for a data segment, namely that the CPL $\geq$ DPL. The gate descriptor, which contains the selector and offset of the destination, is then used to obtain the destination descriptor. If the call is an inter-segment call, CPL must equal DPL, as noted earlier. If the destination code segment is at the same privilege level as CPL, the CS and instruction pointer (IP) registers are loaded from the destination descriptor; note that the calling instruction's offset is ignored, and the entry point specified in the gate descriptor is used instead. If the destination code segment is at a different privilege level than CPL, then DPL must not exceed CPL (outward calls are not allowed.) Otherwise, the new CPL is taken from the destination descriptor. In either case, the CS and IP registers are loaded from the destination descriptor to complete the control transfer.

To maintain system integrity, each privilege level has a separate stack. When the call gate is used to change privilege level, a new stack is selected as determined by the new CPL from the TSS, which has stack pointer information for each of the privilege levels zero through two; there can be no inward call into a destination with DPL equal to three because that is the lowest possible privilege level, so there can have been no less privileged caller. Because a new stack is used, provisions are made to pass parameters from the less privileged stack to the new one; the call gate descriptor includes a count of the number of words that are to be copied from the former (less privileged) stack to the new one. No automatic validation of the caller's access to the words copied onto the stack is done, but instructions are available for the called routine to facilitate the making of such checks. In Trusted XENIX's case, a kernel subroutine performs the necessary access checks whenever invoked.

Interrupt and Trap Gates

Interrupts and traps (exceptions) are special cases of control transfer within a program. Interrupts can be internal (generated by instructions), or external (generated by hardware signals), whereas traps arise when an instruction cannot be completed normally.

The programs used to service an interrupt may execute in the interrupted task's context, or may involve a task switch. The 80286 provides facilities to specify up to 256 different interrupt codes, of which several are

[9] Since privilege levels one and two are not used, only the ring-zero stack is used.

specified by Intel, and the remainder may be determined by the system designer. A gate is associated with each interrupt code, and the descriptors for these gates are held in the variable-length interrupt descriptor table (IDT), which is always pointed to by a special register. The gates in the IDT may be interrupt, trap, or task gates; interrupt and trap gates process interrupts in the same task, while task gates cause a task switch. Interrupt and trap gates are similar to call gates: their access rights field contains only the DPL, and they contain code selectors and offsets for the interrupt routine. The only difference between an interrupt and a trap gate is that, when an interrupt gate is executed, the interrupt-enable flag is turned off, whereas when a trap gate is executed, it is not. Like the call gate, the DPL of the interrupt routine code segment must be numerically no greater than (must not be less privileged than) the interrupted code segment's CPL.

If a task gate is used as the descriptor associated with an interrupt in the IDT, during servicing of the interrupt task switching occurs as described earlier. When the interrupt routine has completed, task switching occurs again with return to the interrupted task. Complete context, including all registers (which includes the CPL), is saved when the interrupt occurs and is restored when the routine exits.

2.5.5 Intel 80386 Characteristics

The Intel 80386 provides a superset of the capabilities that are found in the 80286.[10] The address and data paths of the 80386 are both a full 32-bits. The descriptors for data segments, executable segments, local descriptor tables, and task gates are common to both the 80286 and 80386. However, the 80386 has new versions, in addition to the old ones, of the descriptors for the TSS, call gate, interrupt gate, and trap gate which support the 32-bit nature of the 80386. These new descriptor types are not used in Trusted XENIX.

The 80386 also provides additional registers not found in the 80286, the FS and GS segment registers. New instructions that explicitly operate on the FS and GS registers are available. The addition of FS and GS segment registers must be considered within the Trusted XENIX. When running on the 80386, Trusted XENIX takes explicit action to ensure that these registers are cleared on task switches.

The 80386 also supports the execution of one or more Intel 8086 programs in an 80386 protected-mode environment, called a Virtual-8086 task.[11] However, Trusted XENIX does not provide the capability to enter this mode.

2.5.6 I/O Devices

Disks

Each hardware base in Trusted XENIX supports one or more fixed disk drives (either ST506, ESDI, or SCSI depending on the hardware base), starting at 20 MB in capacity, and one or more diskette drives, with either 1.2 MB or 1.44 MB storage capacity depending on whether a dual-sided 5.25 or 3.5 inch diskette is available.

[10] LOADALL is not supported in the 80386.

[11] Please refer to [386 86] for a more complete description of the V86 mode.

Cartridge Tape Unit

A cartridge tape unit is available to enable both administrators and users to back up large amounts of data quickly. The unit uses one-quarter-inch cartridges and depending on the model of drive, can store 125 MB or 150 MB of data per tape. This unit is accessed identically to the floppy diskette drive. Adaptors are available for both the IBM PC/AT and IBM PS/2.

Keyboard

The keyboard controller is a single-chip microcomputer that is programmed to support the keyboard serial interface. The controller receives serial data from the keyboard, checks the parity of the data, translates scan codes (the codes assigned to each key position on the keyboard), and presents the data to the system as a byte of data in its output buffer. The controller interrupts the system when data are placed in its output buffer. The status register contains bits that indicate whether an error was detected while receiving the data. Data may be sent to the keyboard by writing to the keyboard controller's input buffer. The keyboard is required to acknowledge all data transmissions.

The keyboard sends data in a serial format using an 11-bit frame. Data sent are synchronized by a clock supplied by the keyboard. At the end of a transmission, the keyboard controller disables the interface until the system accepts the byte. Retransmission of bytes with incorrect parity is handled automatically by the controller. The controller also detects excessive delay during a transmission, and indicates an error.

Some of the evaluated hardware platforms implement a keyboard locking mechanism. In the case of the IBM PS/2 machines, there is a power-on password which is handled by BIOS rather than the keyboard controller itself.

The Zenith Z-386 implements a similar locking mechanism, but in the controller itself. The power-on password is stored in a special chip, as opposed to the CMOS RAM, that is accessible by the keyboard controller. When booting, BIOS instructs the keyboard controller to continually look for a matching password and carriage return. During this time, the keyboard will not propogate (beyond the controller) any keystrokes. The advantage of this mechanism is that the machine can perform the entire booting sequence whether the password has been entered or not, but the machine will not respond until the password has been entered.

The AST 386/25 can be configured not to boot until the power-on password is entered. This password is compared against one stored, by BIOS, in a special chip, much like the Zenith Z-386 except that the boot process will suspend until the correct password is entered. Subsequently, the keyboard recognizes a three-key signal which will cause the keyboard to lock; this is known as server mode. The machine will continue to operate normally, hence processes will run to completion and terminals connected via the RS-232 ports can operate freely, but the keyboard will propogate no keystrokes. In order to unlock the keyboard, the keyboard controller must recieve the correct password followed by a carriage return, at which time it will begin propogating keystrokes once more.

No other evaluated machine implements such a mechanism.

Printers

Trusted XENIX provides filtering of output sent to the printer (see section 2.8.8, page 69). This permits a number of printers to be included in the evaluated configuration, since page labeling can not be subverted by software reconfiguration commands contained in user files. Examples of acceptable printers which were tested are listed in Appendix A. Essentially any printer that simply prints (as opposed to interpreting the characters as commands of some sort) the hex characters 00 (null), 08 (backspace), 09 (horizontal tab), 0A (new line), 0C (form feed), 0D (carriage return), and 20 through 7E (standard ASCII characters) is acceptable.

Serial and Parallel Interfaces

The system comes installed with a Serial/Parallel Adapter.[12] The serial portion is fully programmable, and supports asynchronous communications. It will add and remove start, stop and parity bits. A programmable baud-rate generator allows operation from 50 to 9600 baud. Multiple choices are available for character width and stop bit selection. The adapter uses a 9-pin port to which a special cable can be attached which converts the interface to a standard EIA RS-232C port. The serial port is addressed as either communications port 1 or 2, as selected by a jumper located on the system board. The adapter makes all of the accessible registers of the serial controller available for programmable access.

The parallel portion of the adapter provides an eight-bit parallel interface at standard TTL levels; it uses a DB-25 connector. The port may be addressed as parallel port 1 or 2, as selected by a jumper located on the system board. This port is typically used to connect a printer to the system, and uses standard signaling protocol.

2.5.7 System Board Firmware

The ROM on the system board contains the Basic I/O System (BIOS), the BASIC interpreter,[13] and the power-on self-test (see section 2.9.1, page 72). The BIOS provides low-level control for the major I/O devices in the system and is used in Trusted XENIX only as part of the initialization sequence. The BASIC interpreter is a ROM-based interpreter for the BASIC language; it is not used (nor accessible) at all in Trusted XENIX.

2.5.8 Physical Protection

Only the Trusted System Programmer (TSP) is intended to have access to the interior of the system unit. Access to the interior is needed in order to install or repair the system or one of its components, and in the case of a IBM PC/AT to set the few configuration switches that exist, and, finally, to convert the system for operation in maintenance mode.

The Trusted Facility Manual describes this threat and offers suggestions with regard to how each machine can be protected, given their individual physical protection mechanisms. Some of the machanisms implemented

[12] In the IBM PS/2, the serial and parallel ports are on separate boards but equivalent to the IBM PC/AT implementations of the same name.

[13] The BASIC interpreter is included only with some of the hardware bases (e.g., IBM PC/AT).

on the various machines are as described in table 2.1. Even when there is not a power-on password, booting

Hardware Base	**Power-on Password**	**Cabinet Lock**	**Keyboard Lock**
IBM PC/AT	No	Yes†	Yes†
IBM PS/2	Yes	Yes	No
AST 386/25	Yes	Yes	Yes‡
GRiDCASE 1537	No	No	No
NEC PowerMate/BusinessMate	No	Yes†	Yes†
Unisys Personal Workstation *2*	No	Yes	Yes
Zenith Z-386	Yes	Yes	No

† - This is a combined cabinet/keyboard lock.
‡ - This lock is electronic rather than physical.

Table 2.1. Physical Protection Mechanisms

can be protected by physically disabling the `A` floppy diskette drive.

It is the TSPs responsibility to, with the guidance in the TFM, configure and protect the Trusted Product in order to meet the TCSEC assumptions that the TCB portions of the hardware base will not be tampered with.

2.5.9 Hardware Diagnostics

Trusted XENIX provides a number of tools for validation of the correct operation of the TCB hardware. These tools consist of Power On Start Tests (POST), advanced diagnostics, and hardware protection mechanism diagnostics.

The POST resides in the ROM of each evaluated hardware base. Each POST consists of some set of tests that are executed each time the machine is powered-on. The POSTs contain diagnostics to test some minimal functionality of the hardware and the identify available peripherals. The tests include routines for testing the system board, memory and various control units. Additionally, there are tests that verify the correct checksum for the ROM program itself; tests of the registers and flags of the CPU; and some tests of the security features of the CPU. A listing of the ROM program, including the POST, is contained in the IBM PC/AT Technical Reference Manual [TECH].

Advanced Diagnostic Tests are also provided with some of the evaluated hardware bases. These can be invoked only by the TSP, while the system is in maintenance mode. These programs are menu driven and provide more detailed tests, specifically for some of the peripheral devices.

TIS provides a set of diagnostic tests with Trusted XENIX. These tests were developed by TIS to test all of the security mechanisms of the Intel 80286 and 80386 upon which Trusted XENIX depends (e.g., to protect itself from tampering). As with the Advanced Diagnostic Tests, these tests must be run by the TSP from a DOS prompt. This set of tests covers the following areas of security:

- Tests to ensure that Protected Mode can be initialized
- Tests to ensure that privileged and trusted instructions cannot be executed from ring 3

- Tests to ensure that segments can be accessed only with the proper access mode (e.g., read-only segment can only be read)
- Tests to ensure that segments in more privileged rings cannot be loaded
- Tests to ensure that data outside a segment's bounds cannot be accessed
- Tests to ensure that the address validation instructions work properly
- Tests to ensure that invalid descriptors (and, therefore, segments) cannot be loaded
- Tests to ensure that invalid instructions are handle properly
- Tests to ensure that the correct interrupt handlers are called
- Tests to ensure that control transfers to different rings are handled properly and that inproper transfers are not allowed

A supplemental, but necessary (for all non-IBM platforms), set of peripheral tests has also been identified, Check$\surd$It version 3.0 (produced by Touchstone Software Corporation). This utility provides a very comprehensive set of tests that will validate the correct functionality of the evaluated peripherals. This set of tests is necessary to fill in for inadequacies or unknowns in the POSTs of the non-IBM hardware platforms.

2.6 Trusted XENIX Software

This section of the report describes the TCB software. After a general introduction to the classes of software included in the TCB, the kernel software is described, followed by a more detailed description of the file system and process management. The kernel interface is then described, followed by the trusted processes.

2.6.1 Classes of TCB Software

Trusted XENIX's Intel 80286 or 80386 CPU provides two addressing modes (see section 2.5.4, page 12) of operation: real mode and protected mode. While running, Trusted XENIX always uses protected addressing mode. The processor's real address mode is never used except in the earliest stages of system initialization and at the end of system shutdown. Likewise, the standard IBM PC/AT, IBM PS/2, or PC/AT Clone ROM BIOS firmware is never used except at those times, because it operates only in real address mode.

The protected mode provides four privilege levels, numbered zero through three. The kernel runs entirely in the processor's most privileged state, privilege level zero. This allows it access to the privileged instructions that control I/O operations and the address translation tables. Kernel software is defined as all that which runs with this hardware-defined privilege. Most kernel software runs on behalf of some process and is invoked by system calls or emulation service requests. The remainder is used to perform initialization and handle hardware interrupts. No software in Trusted XENIX uses privilege levels one or two.

All non-kernel software runs as part of some process (trusted or untrusted), and runs in the least privileged hardware state, privilege level three. A *trusted process* (TP) is any process responsible for maintaining the secure state of the system, or which has the *capability* to affect the secure state of the system (even though it may be trusted only not to exercise that capability or privilege). In other words, any process which has privilege not granted to ordinary user processes, or on which the secure operation of the system depends, is a trusted process.

The TCB is written primarily in the C language, and uses a subset of the standard C program library. All the TP code is written in C, but the kernel contains significant amounts of 80286 assembler language code. Some of the assembler code is directly security-relevant (generally, the code which deals with memory management, traps, and interrupts, and small parts of I/O drivers). The bulk of the assembler code is used to provide emulation for the instructions which would otherwise have been implemented by the Intel 80287 or 80387 Numeric Processor Extension, when such a processor is not present in the hardware configuration.

2.6.2 Kernel Software

The Trusted XENIX kernel is organized into 10 conceptual subsystems which are summarized in table 2.2.

Subsystem	**Description**
Configuration Subsystem	Defines size of system tables, allocates space for some system data structures.
Initialization Subsystem	Load system during boot, invokes *swapper* and *init*.
System Call Subsystem	Validates system call arguments, and calls appropriate handler in kernel.
Security Subsystem	Responsible for all security decisions in the system. Implements the Reference Monitor.
Process Subsystem	Primarily responsible for swapping and context switching for processes.
Memory Subsystem	Manages address spaces, system memory, and the swap space.
Input/Output (I/O) Subsystem	Contains functions for implementing I/O in the system, including all device drivers.
File Subsystem	Implements the XENIX file system and associated functions (e.g., pathname resolution, inode management).
Inter-Process Communication (IPC) Subsystem	Implements IPC mechanisms: semaphores, shared segments, and message queues.
Miscellaneous Functions Subsystem	Contains routines which do not fit in to any of the other subsystems.

Table 2.2. Trusted XENIX kernel subsystems

Each of these conceptual subsystems is implemented by a collection of source code and configuration tables which are partitioned among 13 source directories.

This section summarizes the important aspects of each subsystem. The major functional responsibilities of the subsystem are described, followed by a summary of its security-relevant activities. When a subsystem implements a particularly important security mechanism, that mechanism is referenced and described in detail later.

Configuration Information

This subsystem contains the two data files which define the size of system tables, various system data areas, and any options specified when the Trusted XENIX kernel is built. These files contain source code statements which reserve space for some system global variables. These files do not directly specify any security-relevant information, but do contain information that has a bearing on the correct operation of the system (for instance, the number of entries in the process table).

Initialization Subsystem

This subsystem is responsible for initial loading of the system, initialization of hardware devices, interrupts, and address space tables. Its final act is to start the *swapper* and *init* trusted processes (see section 2.6.6, page 35). It has no security responsibilities except for correct initialization of the system. The initialization subsystem is the only Trusted XENIX code which ever runs in real addressing mode: during the earliest stages of initialization, it switches from real to protected mode, and the system continues to run exclusively in protected mode until the very end of the shutdown process.

System Call Subsystem

This subsystem contains the functions which receive control when a process issues a system call to the kernel's call gate entry point. It is responsible for argument checking and various housekeeping tasks and for selecting the appropriate function elsewhere in the kernel to carry out the system call. Its security responsibilities include much of the work of auditing and kernel argument validation (see section 2.6.5, page 31). It is also partly responsible for general kernel isolation.

Security Subsystem

This represents the Reference Validation Mechanism which implements the Reference Monitor of Trusted XENIX. It is a collection of functions called from elsewhere in the kernel to make access checks and implement the Discretionary, Mandatory, Privilege, and Auditing policies. This is where actual interpretation of security data (labels, permission bits, Access Control Lists (ACLs), privilege vectors) is performed, and where audit messages are written to the audit file. It also implements the ACL system calls. Its sole responsibility is security.

Process Subsystem

This subsystem contains the functions responsible for process multiplexing (via swapping) of in-memory processes, and context switching among processes. It is also responsible for initial handling of all hardware interrupts, management of the clock, and the sleep/wakeup functions used for synchronization within the kernel. Finally, it includes the user interfaces (the *kill*, *signal*, and *ptrace* system calls) for inter-process signals and process tracing. Its primary security responsibility is control of signals and other interactions between processes: maintaining the process isolation required to meet the B2 System Architecture criterion. A detailed description of process management appears later (see section 2.6.4, page 28).

Memory Subsystem

This subsystem contains the functions responsible for managing process address spaces, the system's real memory, and the swap space. It includes the system swap process, which is responsible for making all decisions on when to swap a process to or from secondary storage. It also includes the system call handler routines for process memory management and is responsible for loading executable files into a process address space as part of the *exec*[14] system call. Its major security responsibility stems from its memory management functions, which include functions relating to Object Reuse, access control, kernel and process isolation, and maintenance of the (hardware-interpreted) table that controls the operation of the hardware portion (see section 2.5.4, page 13) of the Reference Validation Mechanism. It is also responsible for management of privileges and setuid/setgid changes during the *exec* system call.

I/O Subsystem

This subsystem is responsible for actual I/O operations within the kernel. All other subsystems call the I/O subsystem to perform I/O, either going through the buffer cache manager for block device I/O, or through the line discipline and character list managers for character I/O. The I/O driver routines are called to initiate and manage the physical transfer. No security decisions are made by this subsystem, but the character I/O portion is responsible for implementing the Secure Attention Key (SAK) mechanism (see section 2.8.6, page 68).

The standard Trusted XENIX device drivers for physical devices include: disk I/O (fixed disk and diskette), tape I/O, serial communications adapter, printer (parallel) adapter, keyboard, and graphics (monochrome and color). The keyboard driver is responsible for interpreting the "press/release" codes delivered by the keyboard hardware interface and translating these to ASCII codes. The keyboard driver interacts directly with the microcontroller firmware in the keyboard, and does not use any of the IBM PC/AT, IBM PS/2, or PC/AT Clone's ROM BIOS routines. The graphics drivers, similarly, interact directly with the appropriate graphics controller hardware, translating ASCII codes to their screen representations, and do not use the ROM BIOS routines.

In addition, there are drivers for "logical" devices. Two of these (*/dev/null* and */dev/tty*) represent "security-eccentric" devices that are always accessible to all processes for both reading and writing (without regard to mandatory access controls), either because they do not represent storage objects[15] (*null*) or because they are simply a form of indirect reference to a real device which does obey mandatory access controls (*tty*). Other logical devices (such as */dev/mem*, */dev/kmem*, and */dev/swap*) provide access to internal TCB data structures and are accessible only to appropriate trusted processes.

File Subsystem

This subsystem includes all the functions that implement the Trusted XENIX file system, above the level of basic buffered I/O. This includes pathname resolution, directory management, inode management, access control, file locking, file I/O, file and inode space allocation, pipe handling, filesystem mounting, and hidden

[14] There are a number of routines a process can use to execute a program, although there is only one entry point into the kernel (*execve*) which is where all these routines eventually go. In keeping with common UNIX literature, these routines and the system call they employ will be referred to collectively as the *exec* system call throughout this report.

[15] For more on storage objects, see section 2.7.2 on page 45.

subdirectory deflection (see section 2.8.9, page 70). Most of this subsystem is directly concerned with responding to system calls manipulating user-visible objects. Its main security responsibility is invocation of the Reference Validation Mechanism discretionary and mandatory access controls on all file system objects. The pathname resolution part of the file subsystem is responsible for some of the access checks and for implementing hidden subdirectories.

Interprocess Communication Subsystem

This subsystem implements the user-visible inter-process communication (IPC) objects: XENIX Semaphores, System V Semaphores, XENIX Shared Memory Segments, System V Shared Memory Segments, and System V Message Queues. Its functions are all directly related to providing the system call interface for these objects. Its major security responsibility is invocation of the Reference Validation Mechanism for discretionary and mandatory access controls on IPC objects. It is also responsible for enforcing Object Reuse controls on these objects.

Miscellaneous Functions

This subsystem contains some miscellaneous kernel routines that do not belong with any particular subsystem: shutdown, kernel debugging support (disabled in Trusted XENIX), kernel performance monitoring, random number generation, floating point emulation, assorted assembler utilities, and miscellaneous system calls. They have no security responsibilities.

Header Files

Although not identified as a subsystem in the same sense as the other subsystems just described, the collection of files which declare global data, types, and structures is important and needs to be discussed. The header files consist of collections of global structures and variables. When a header file is `#include`d in a .c source file (or in another header file), then all of the structures and variables declared in that header file are available to the `#include`ing file. This is the method used to pass global variables in the system. Each Trusted XENIX module `#include`s only those header files which have some combination of variables, structure definitions, static data (e.g., those elements which are `#define`d), and macros it uses. Each header file contains comments identifying the subsystem to which each variable and structure "belongs." A large number of data declarations may appear in each header file, and all of these are accessible to a source file which `#include`s it, even though only one declaration may be actually used by the `#include`ing module.

2.6.3 File System

The filesystem is a collection of information stored on disk. The filesystem is managed by the kernel, which uses a number of data structures (e.g., superblock, inode list, and directory) to store the necessary control information.

The disk is divided (by the TSP) into a number of partitions (maximum of 4) in which filesystems reside. The disk partition table, which resides in the first partition, defines which areas of the disk are reserved for which partitions. A partition's type can be system, for root directory files, or user, for programs and files of the operating system.

The installation of Trusted XENIX removes already existing partitions from the disk. Although it is possible to store different operating systems (e.g., DOS) in different partitions, this capability is not supported. The TSP is instructed in the TFM to ensure that DOS partitions (if there are any) are not marked bootable. A filesystem is created on a partition via the *fs*, *device*, and *mkfs* commands.

A filesystem can be viewed at the highest level as beginning with a boot block, followed by the super block, which is followed by the inode list, which is followed by a collection of data blocks.

The following will briefly describe each of the important data structures and how each supports the filesystem.

- **The boot block**: The boot block is the first sector of every filesystem. The boot block usually contains the bootstrap code that is read into the machine to initialize the system. While the system only requires one boot block, every filesystem must have a boot block. As a result, it is possible for the boot block to be empty.

- **The superblock**: The superblock is stored beginning in the first sector after the boot block. The superblock contains the high-level information necessary to describe the filesystem. For example, the superblock contains a pointer to the root of the filesystem, the size of the filesystem, and pointers to the free lists, the next available file entry and next available data space.

- **The inode list**: The inode list is stored beginning immediately after the superblock. The inode list is simply a list of inodes where each inode is referred to by its index in this list. This index is known as its inode number. An inode is the fundamental filesystem description of a file (and other objects with a filesystem representation) and its attributes. An inode contains file type information (e.g., block or character special, file, directory), access control information (e.g., file owner's uid and gid, protection bits), status information (e.g., date/time file was last used, date/time file was last modified), number of links to the file, file size, and pointers to the disk addresses of the data blocks that comprise the file. Also in the inode as one of the permission bits is the "sticky bit" (settable by TSP and SSA). The kernel does not release memory allocated to a file which has its sticky bit set.

- **Data blocks**: The data blocks are the containers for the information that is to be stored in the filesystem. Any particular data block may be associated with only one inode; however, many data blocks may be associated with a single inode.

The main memory data structures which support the filesystem include a per-user open file table. This table is indexed by file descriptors and its entries point to the entries in the system file table, where an entry for each open file is maintained. The system file table entry includes items such as open flags, seek cursor and a pointer to the system inode table. Entries in the system inode table are copies of the on-disk inode (with some additions such as status information, logical device number of containing filesystem, referent count, etc) for a file in use. Associated with a main memory inode is the lock list (singularly linked list of locked regions) for that inode. All lock lists are kept in a Lock Table. The system periodically executes the *update* routine to write main memory data structures (superblocks, inodes, and disk blocks from the block buffer cache) to disk.

Mounting File Systems

Filesystems (stored on disk partitions) have an assigned minimum and maximum security label, and can be mounted and dismounted. Users who can run the necessary programs installed with the MOUNT privilege can mount and unmount filesystems.

The kernel maintains a mount table with entries for all the mounted filesystems. The Secure Devices table (*s_device*) contains device ranges which are checked by the *mount* TP against the security label range of the filesystem (recorded in the superblock). The *mount* TP also rejects any filesystem not marked as a TIS Trusted XENIX filesystem in the superblock and creates a */.ACL* subdirectory in the mounted filesystem's root directory if the mount is write-enabled and the subdirectory does not exist. The Secure File Systems Table (*s_fs*), which specifies filesystem mount points, contains a mountability flag for each mount point. This flag determines what type of user (ordinary, TSP, SO) may perform the mount.

A TSP may mount/unmount any filesystem. An SO may mount/unmount all filesystems that are not marked "TSP-mountable". Ordinary users may mount/unmount filesystems marked "user-mountable" provided they have write access to the *s_fs* table-specified mount point. The TSP and SO use the *mount* and *umount* commands to mount and unmount filesystems. As shipped, the system has this privileged command available only in the SO privileged trusted shell. However the TSP can change the Command Table (*s_cmd*) to make the *mount* command available to ordinary users (for displaying the mount table only) in the ordinary user restricted shell.

Ordinary users must use the *usrmnt* command to mount filesystems. The *usrmnt* command invokes the *usrmnt* TP, which in turn will use the *mount* system call. However before calling *mount*, *usrmnt* checks for any privileged or setuid/setgid files residing on the filesystem. If the filesystem has these files residing on it and is to be mounted read-only, the *mount* is rejected. If the filesystem has these files residing on it and is to be mounted write-enabled, all file privileges and setuid/setgid bits on that filesystem are stripped before the filesystem is mounted.

2.6.4 Process Management

The process subsystem is responsible for creating a multitasking environment. It supports the concurrent processing facilities such as *fork* (create a "child" process) and *wait* (wait for a child process to terminate). In addition, it maintains system time and date, provides signal, process tracing, and time event services, handles hardware interrupts and traps, and manages the processor.

The process subsystem comprises the following seven distinct modules, of which the most significant are the processor manager and the process manager:

- First level interrupt and trap manager
- Signal manager
- Second level trap manager
- Second level interrupt manager
- Time manager
- Processor manager

Execution Mode	**Description**
User	Executable Code (from program file)
User	Stack
User	Heap
User	Shared segments (optional)
Kernel	Swappable Process Context (`u_area`)
Kernel	Local Descriptor Table
Kernel	Stack
Kernel	Global Data (per-system)
Kernel	Executable Code

Table 2.3. Process Layout

- Process manager

The processor manager is responsible for creating a virtual processor environment for each process by multiplexing the one real processor among the various processes.

The process manager module maintains the internal representation of a process, supports the Trusted XENIX concurrent processing facilities such as *fork* and *wait*, and manages the attributes associated with a process. Each process is associated with a unique identifier, called the process ID (pid), and some attributes as described in section 2.7.1, page 42.

A process can be executing user code or kernel code—distinctions recognized in hardware via the privilege level mechanism. When it is executing user code, it is in the user mode. When it executes a system call that requires the execution of some kernel code, the process changes to kernel mode. The single gate entry point to the Trusted XENIX kernel is an Intel 80286 or 80386 call gate located in the GDT. When the gate is called, the CPU switches modes from privilege level three (user mode) to privilege level zero (kernel mode), switches to the kernel stack, and enters a subroutine called *kentry*.

Even if a process makes no specific requests for operating system services, the operating system will be invoked to perform bookkeeping operations that relate to the user process: handling interrupts, scheduling processes, managing memory, and so on. Processes in user mode can access their own instructions and data, and other processes' instructions (for shared text segments) but not kernel instructions and data. Processes in kernel mode can access both kernel and user addresses. This protection is provided by the hardware.

Although the system executes in one of two modes, the kernel runs on behalf of a user process. The kernel is not a separate set of processes running in parallel to user processes, but rather it is a part of each user process.

The system hardware is aware only of the distinction between the user mode and the kernel mode; it cannot distinguish between instructions from different processes or among users executing these processes. The operating system keeps internal records to distinguish the many processes executing on the system.

Process Address Space

The address space of a process is structured into parts represented by one or more Intel 80286 or 80386 segments, as shown in table 2.3. The user executable code may be a single segment for small model programs, or multiple segments for medium, large, and huge model[16] programs. The heap may be multiple segments for large and huge model programs. Shared segments are represented by one or more hardware segments for each segment (System V Shared Memory or XENIX Shared Memory) being shared. The kernel areas for the `u_area`, LDT, and the stack all share the same hardware segment. The kernel executable code is multiple segments because the kernel is a medium model program. The kernel executable code also includes the kernel's call gate segment (see section 2.6.5, page 31).

The user-mode parts of an address space are initialized from the program file specified in an *exec* system call. The size of the stack and heap segments may be changed with the *sbrk* system call. Shared segments may be added using system calls (e.g., *shmget* and *shmop*). The kernel-mode parts of the address space are initialized during system initialization and process creation. For the most part, the kernel-mode parts are inherited by all processes at creation time and are modified only slightly.

The LDT of the address space is the hardware-interpreted table that defines the per-process address space (the user-mode parts). Additionally, a system-wide shared table known as the GDT describes the kernel-mode address space. Because some parts of the kernel-mode elements are actually per-process (the stack, LDT, and `u_area`), these parts are copied in and out of the system-wide (as defined in the GDT) segment for kernel process context at every process context switch.

Logically, the stack, LDT, and `u_area` contain similar information: process context that must be addressable and in main memory only when the process is running. They contain kernel information required only when the kernel is running (when invoked by a system call, for instance) on behalf of that particular process. Other process context, which may be required when performing kernel services in another process or when handling interrupts, is in the process' `proc` structure, which is kept in kernel memory in a permanently allocated table accessible to all processes.

Interrupts and Signals

Trusted XENIX allows devices such as I/O peripherals or the system clock (interval timer) to asynchronously "interrupt" a process. On receipt of the interrupt, the kernel saves the current context (image) of the process, determines the cause of the interrupt, and services the interrupt. The kernel is able to prevent the occurrence of some interrupts during critical activity; principally those which could corrupt data if interrupts were allowed. Traps have a similar effect on a process but are the result of unexpected events caused by the process, such as addressing illegal memory, executing privileged instructions, or dividing by zero.

Analogous to the interrupt and trap mechanisms in hardware is a Trusted XENIX facility called *signal*, which is implemented in software. Signals provide a mechanism for the course of one user process to be interrupted, diverted, or terminated by the action of another process or as the result of an error or terminal operator action. The kernel handles signals in the context of the user process that receives them, so a process must run to receive them. Trusted XENIX handles signals in three ways: the process exits on receipt of the signal, it ignores the signal, or it executes a particular (user) function on receipt of the signal. Access control checks for allowing signals between processes include discretionary, mandatory, and privilege checks (see section 2.7.2, page 55).

[16]The term "model" refers to the type of addresses (pointers) which can be used within a program and is an artifact of the Intel 80286 or 80386's 65,536 byte segment size. The small model uses only one segment each for code and stack, the medium model uses multiple segments for code and one for stack, and a large model program uses multiple segments for each. A huge model program is a Trusted XENIX concept meaning a process whose segments are large enough to fit in memory, but too large to be swapped to the swap space on disk. Ordinary users can not create huge model processes on Trusted XENIX.

2.6.5 Kernel Interface

The Trusted XENIX kernel interface is provided by system calls, and, if the system does not include an Intel 80287 or 80387 Numeric Processor Extension, by hardware-detected exceptions requesting emulation of the 80287's or 80387's instructions. Additionally, the kernel is invoked whenever any hardware-detected invalid operation (such as an invalid address, invalid opcode, or overflow) occurs, although this is not the usual way to request a kernel service, and normally results in process termination.

Floating point emulation requests are received by the hardware interrupt handler and sent on directly to the floating point emulator code in the kernel. They never cause I/O or otherwise interact with the rest of the kernel and are of no further interest here.

Ordinary system calls are made by a `CALL` instruction referencing the only call gate in the Trusted XENIX address space. The process switches from privilege level 3 (user mode) to privilege level 0 (kernel mode) and transfers to the assembler function *kentry*. This saves some registers and prepares the ring 0 calling environment, then calls the C function *scall*. In *scall*, the process environment is consulted to determine which system call table to use (this depends on what type of program file the process is executing: Version 7, System III, or System V; this information is loaded into the process environment when the program is *exec*ed), and an entry is selected from the table based on the system call number found in the processor's `AX` register on entry to kernel mode. This table entry specifies the number of arguments and their types; this information is used to copy the arguments from registers or from the user stack into the kernel's argument storage area. Different functions are used for copying and validating arguments depending on the memory model in use by the calling process' program file; however, all arguments are kept in the same canonical format once copied into the kernel.

Once arguments are copied, auditing setup (if required) is done, an abnormal exit handler is set up, and the system call service routine is called. When it returns, or an abnormal exit (via *longjump*) occurs, *scall* regains control, copies the return arguments back to user space, and performs any necessary post-processing, such as generating final audit messages, rescheduling the process, and restoring parts of process context. Finally, it returns to *kentry*, which restores registers and returns to user mode.

In system call service routines, parameters are referenced through a structure pointed to by the process global variable `u.u_ap`. By the time the service routine is called, all scalar parameters have been copied into kernel storage, and are referenced directly. Because the kernel copy is protected from modification once copied, scalar parameters are inherently safe from multiple reference problems–and access to parameters was already ensured by the checks made when *scall* copied them into kernel space.

For pointer parameters, such as character strings or arrays, the pointer itself has been copied into kernel storage, but the data pointed to have not been. Therefore, each system call routine must copy the data itself, ensuring that they are accessible to the user process. The system call routines must also be explicitly coded to avoid multiple references to user parameter data. Because UNIX started out as a system where direct parameter references were impossible, Trusted XENIX inherited a well-defined programming discipline for copying data between kernel and user space: the *copyin* and *copyout* routines. These TCB routines make the appropriate access checks, so that the system call routines themselves need not be concerned about invalid values in user-supplied pointers. These routines use the Intel 80286 or 80386-provided hardware instructions for parameter validation.

Multiple parameter references are avoided by programming convention, again a fairly centralized one. Because most pointer parameters are used to point to pathnames, and the standard mechanism for examining pathname parameters processes the user data one byte at a time, this nearly guarantees that pathname parameters are referenced precisely once in user space. To re-use the data from user space (that is, copy and interpret exactly the same data residing in user space) rather than making a copy or otherwise interpreting it the first time it is referenced in the kernel and saving the result would be awkward and inefficient. Other pointer parameters are simply copied in their entirety, exactly once, near the beginning of the routines that reference them. These protections are provided by programming convention and system structure, rather than by any actual mechanism.

That these programming conventions be followed in all system calls is critical to the security of the system. Because direct references through user-supplied pointers work in Trusted XENIX's Intel 80286 or 80386-based implementation (that is, they do not fault), no automatic detection for a missing call to *copyin* or *copyout* exists, as might be found in systems with less sophisticated memory architectures. Similarly, no automatic mechanism for detecting multiple references to user parameters exists. Enforcement of these conventions is assured by the Configuration Management system's code review procedures (see section 2.9.2).

The single interface point (*scall*) with its table-driven interpreter and parameter copying mechanism encourages simple and uniform interfaces to system calls. This, in turn, can reduce the complexity of system call handlers, and thus, of the TCB itself. Although the Intel 80286 or 80386 architecture would permit more complex and/or efficient interfaces, Trusted XENIX does not use those hardware features. The uniform interface also simplifies auditing and processing of error returns.

2.6.6 Trusted Processes

This section describes the role of Trusted Processes (TPs) within Trusted XENIX and the specific privileges and capabilities of each TP. The following types of privilege are provided:

Special user and group identity (see section 2.3.2, page 6): Processes running with these special identities have *"discretionary"* access to administrative files and directories not accessible to ordinary users. This privilege is implemented using Trusted XENIX DAC and user identity mechanisms, by assigning special uids and gids to certain user and group identities.

This privilege type is used primarily to partition the administrative databases and make them accessible only to the TPs which manage them. The special identity may be administratively assigned (as is the case for "permanent" TPs) or dynamically acquired through use of the setuid/setgid mechanism (see section 2.8.1, page 57), which can change a process' uid based on attributes of the process' executable program file or through use of the *setuid* system call.

Generalized Privilege Mechanism (GPM) (see section 2.8.3,page 60): Processes running with GPM privileges are permitted to invoke privileged system calls and privileged options of other system calls. The GPM privileges are always acquired dynamically based on attributes of a TP's program file (much like acquiring a special uid through the setuid/setgid mechanism). Each GPM privilege[17] corresponds to one specific

[17]There are 36, and a TP may be given any subset of them.

privileged operation, so that a TP may be restricted to performing only those operations necessary for its correct functioning. Additionally, the GPM privileges may be acquired and dropped dynamically during the operation of a TP, so that privileged operations can be explicitly limited to small regions of code. This is particularly important for the privileges that provide exemption from access control (MAC_EXEMPT and DAC_EXEMPT).

Trusted XENIX differs from most UNIX and UNIX-like systems in that uid 0 (*root*) has no special privileges except for access to *root*-owned files. The GPM mechanism entirely replaces the privileges normally granted to *root*-uid processes with regard to system calls and exemption from normal access controls. This is the cornerstone of the system's implementation for Trusted Facility Management and meeting the "least privilege" aspect of the System Architecture requirement. The TSP runs as *root* in single-user mode.

Trusted XENIX contains three types of TPs:

User Invoked: invoked by a command or *exec* system call from an unprivileged user or process, not involving the trusted path (see section 2.8.6, page 68). These are used for functions which do not require a trusted user interface, but which do require that the program performing the operation have some privilege, which may be either GPM privileges or a special identity. Examples include *mkdir* and *lp*.

Trusted Shell: invoked by a command issued to the trusted shell (*tsh*). The trusted shell is invoked by establishing a Trusted Path[18] using the Secure Attention Key (SAK) mechanism (see section 2.8.6, page 68). This places the user in direct communication with the trusted shell, which in turn invokes selected TPs. Some trusted path TPs require no additional privileges (GPM or special uid) to perform their functions, but are trusted to guarantee that the information displayed or operation performed is what the user requested. Others require additional privilege (like the user-invoked TPs) as well as depending on direct communication with the user. Examples of trusted TPs include *ls* (which is not otherwise privileged), *star* (which is privileged), and the administrative and operator interface programs (which require that the user has selected an administrator or operator group ID).[19]

Permanent: created during system initialization or by a privileged user and always running to perform system services. The "permanent" TPs always have the special uid or GPM privileges. Most are created during system initialization, but some (the printer spooler process, for instance) may also be restarted dynamically by privileged users. Examples include *init*, *getty*, *lpsched*, and *swapper*.

Trusted Process Protection Mechanisms

All TPs are coded in a uniform manner to provide greater assurance of correct implementation. Source code modules for all TPs include standard commentary which identifies the TP name, source file name, short

[18] Although *tsh* can be invoked as an ordinary user command, the TPs it invokes will detect this and refuse to exercise their privileges because they were not invoked through the Trusted Path.

[19] Trusted XENIX provides a *restricted trusted shell* for the administrator (or operator) wishing to use administrator (operator) TPs. A separate such restricted trusted shell is provided for each of the special user types, and is invocable only from within the (normal) trusted shell.

description of purpose, and installation data, which includes a list of the privileges given to that TP and the rationale for doing so. This standard commentary is followed in each of the TP source files by code which implements a standard initialization of the TP environment. The routines listed below provide the primary functions used by the standard initialization.

- *chkTPstate()* function: takes as its input parameter a bit vector representing a set of allowed TP states. If the TP state is one of that set, *chkTPstate()* returns that state; otherwise it returns STATE_NULL.
- *close_all_but(arg)* system call: the specified "arg" gives a list of file descriptors which should not be closed. All other file descriptors are closed.
- *umask* command: sets the default mask used to set the DAC protection bits at object creation time.
- *alarm* call: executed, if appropriate, to set/clear the alarm timer.

In addition, each of the TPs depends on some of the security mechanisms listed below. Kernel mechanisms include:

1. The Generalized Privilege Mechanism
2. Mandatory Access Control and Discretionary Access Control mechanisms (except where a trusted process is exempted by the GPM)
3. The Secure Attention Key (SAK)–many TPs may be invoked only by using the trusted path
4. Object Reuse mechanisms
5. Kernel and hardware-provided isolation mechanisms
6. Audit mechanisms

TP-specific mechanisms are:

1. The Power Hierarchy Mechanism, used only by Trusted Processes, defines a hierarchy among administrative users (see section 2.8.3, page 61).
2. The *scheck* TP, which performs integrity and consistency checks on security profiles and all command and data files required for secure operation of the system. It also checks the filesystem to determine whether it satisfies the compatibility rule[20] and checks for the presence of malicious programs (e.g., those that have either of the *setuid/setgid* permission bits set to a non-existent uid/gid or to a pseudo-user/group other than as specified in *s_install*), or those with privileges not in *s_install*.
3. The *chkCPL()* function, which takes the session security level as the input parameter and returns TRUE if Current Process Level matches the specified security level, else returns FALSE.

In Trusted XENIX, all TPs and their related files must be specified in the Installation Table *s_install*. This specifies the complete protection controls required for the secure operation of the system. Only the TSP has access to the *s_install* file. During Trusted XENIX installation, all TPs listed in *s_install* are given the attributes stated within the table entries. These attributes are:

[20]The "compatibility rule" is described in detail on page 25 of *Secure Computer Systems: Unified Exposition and Multics Interpretation* [BELL].

- program pathname
- owner
- group
- mode
- ACL
- security label
- privileges

Trusted XENIX's Trusted Processes

The following tables describe all the TPs in Trusted XENIX. The tables group the TPs into their types, as described above (i.e., user invoked, trusted path, permanent). The table entries show which users may execute the TP, the GPM privileges given to the TP, the effective uid/gid used by the TP, and any additional notes.

For user-invoked TPs, the effective uid and effective gid are set by the setuid/setgid mechanism. For permanent TPs, they may be inherited from the caller. When the effective uid and effective gid are listed as something other than the invoker of the TP, the TP may be assumed to execute with its setuid and/or setgid bits(s) turned "on." It may, in that case, toggle its effective uid (effective gid) between that of the invoker and the uid (gid) specified in the uid (gid) field (see section 2.8.1, page 57).

Permanent TPs[21] are either always running or are dynamically created by processes that are always running. They do not run with their setuid or setgid mode bits turned "on."

[21] Three of the entries in the permanent TP table are not TPs: *swapper* is considered to be part of the kernel, and *cron* and *update*, both of which are initiated during system initialization, do not execute with privileges.

User-Invoked TPs				
TP Name and Function	Privileges	uid	gid	Notes
auditnam: defines initial state of audit parameters file	AUDITLOG, DAC_EXEMPT, MAC_EXEMPT, SETUID	*audit*	*audit*	Can be invoked only by TSP in MAINT_MODE.
c_attr: shows current process attributes	none	*bin*	invoker	
cancel: cancels line printer requests	KILL, MAC_EXEMPT	*lp*	*so*	
chmod: changes mode of files	none	invoker	invoker	
df: reports the number of available disk blocks	none	*sysinfo*	invoker	*sysinfo* is a pseudo-user which owns special disk files containing status information.
dmesg: displays system messages	none	*sysinfo*	invoker	Can be invoked only by TSP in MAINT_MODE, or by members of the cron group.
emkdir: performs setuid/setgid to effective uid/effective gid, then invokes *mkdir*	SETUID	invoker	invoker	
format: formats diskettes	none	invoker	*remmed*	*format* is restricted to the console terminal.
ipcs: reports IPC facility status	none	*sysinfo*	invoker	Users are not allowed to see any information about IPC objects at a higher security level than the CPL.
kill: sends a signal to a running process	AUDITLOG	*bin*	*bin*	
login: gives access to the system	MAC_EXEMPT, DAC_EXEMPT, AUDIT, AUDITLOG, CHOWN, NICE, SETFLBL, SETPRLBL, SETUID, ULIMIT	invoker	invoker	*login* is not actually invoked by a user command but rather by pressing the SAK. It is also executed by the *getty* TP. More detailed information on *login* is given in the overview section on Subjects.
ls: lists the contents of a directory	none	invoker	invoker	*ls* will print out directory entries only at or below the CPL.
lp: sends requests to the line printer system	MAC_EXEMPT, SETFLBL, AUDITLOG	*lp*	*so*	*lp* writes the requested file to the spool directory at the time of the request, operating under the requestor's uid. The *lpsched* daemon notes the request and performs the printing.
lpadmin: configures line printer spooling system	AUDITLOG	*lp*	*so*	Can be invoked only by TSP in maintenance mode.
lpstat: reports status of printer, daemon, and queues	AUDITLOG, MAC_EXEMPT	*lp*	*so*	An unprivileged user may view only his or her own requests at or below their current privilege level. The SO can view all requests.

User-Invoked TPs (cont.)

TP Name and Function	Privileges	uid	gid	Notes
mkdir: creates a directory	DAC_EXEMPT, LINK_DIR, MKNOD, SETFLBL, AUDITLOG	invoker	invoker	Will not create a directory at a level below or incomparable to the parent directory.
mkfs: makes a file system	AUDITLOG	invoker	invoker	Can be invoked only by the TSP in MAINT_MODE.
mv_dir: moves directories	LINK_DIR SIGNAL	invoker	invoker	Trusted not to create loops in directory structure. Can also be invoked by *mv* command.
ps: reports process status	AUDITLOG	*sysinfo*	invoker	Displays status only for processes at or below current level of requester.
rmdir: removes a directory	DAC_EXEMPT, LINK_DIR, MAC_EXEMPT AUDITLOG	invoker	invoker	
slutmp: filter for the file */etc/utmp*	none	*root*	invoker	*/etc/utmp* contains one entry for each terminal in the system. Users may see only those entries that are at or below their CPL.
tsh: trusted shell	none	invoker	invoker	This is a table driven command interpreter which assures communication between the user and the TCB. The table */etc/security/s_cmd* drives the interpreter, and may be edited only by the TSP. Any user may invoke the trusted shell by using the SAK mechanism. Most commands invocable by the trusted shell contain a "guard" which prevents them from executing outside the trusted shell (see section 2.8.6, page 68).
usrmnt: mounts removable-media file system	AUDITLOG, CHMOD, CHOWN, DAC_EXEMPT, MAC_EXEMPT, MOUNT, SETFLBL, SETPRLBL, SIGNAL	invoker	invoker	

Trusted Shell TPs					
TP Name and Function	Run by	Privileges	uid	gid	Notes
accept: allows printer requests	SO	MAC_EXEMPT, AUDITLOG	*lp*	*so*	
accton: enables or disables process accounting	AA	ACCT, AUDITLOG	invoker	invoker	
acl: ACL operations	SSA	ACL, AUDITING, DAC_EXEMPT, MAC_EXEMPT	invoker	invoker	SSA executes *acl* from the privileged menu of the trusted shell. Privileges are applied only in this case. Unprivileged users execute *acl* as an ordinary command.
audit_star: allows auditor to invoke *star*	AUDIT	AUDITLOG, SETUID	invoker	invoker	Can be invoked only from the Auditor menu of the privileged trusted shell.
auditsh: performs security audit trail operations	AUDIT	AUDIT, AUDITLOG	*audit*	*audit*	Can also be invoked by */etc/rc* at system initialization.
c_chaudlvl: changes audit level of user/group	AUDIT	AUDITLOG, SETPRLBL	*bin*	invoker	*bin* is a pseudo-user which owns many of the system files
c_chkey: displays and/or changes the *star* encryption key	SSA	MAC_EXEMPT, AUDITLOG	*bin*	invoker	SSA may use other untrusted processes to read (only) the *star* key.
c_chlabel: changes a file security label	SSA	DAC_EXEMPT, MAC_EXEMPT, SETFLBL, AUDITLOG	invoker	invoker	
c_chmod: changes file mode (including setuid and setgid bits)	any user	DAC_EXEMPT, MAC_EXEMPT, CHMOD, AUDITLOG	invoker	invoker	
c_chowner: changes file owner and group	SSA	DAC_EXEMPT, MAC_EXEMPT, CHOWN, CHMOD, AUDITLOG	invoker	invoker	

Trusted Shell TPs (cont.)					
TP Name and Function	Run by	Privileges	uid	gid	Notes
c_fixlabel: sets file label equal to label of containing directory (i.e., "fix")	SSA	AUDITLOG, DAC_EXEMPT, MAC_EXEMPT, SETFLBL	invoker	invoker	
c_halt: halts the system	any user	SHUTDN, DAC_EXEMPT, AUDITLOG	invoker	invoker	Must be invoked from the system console.
c_kill: kills any process (except process 1)	SSA	AUDITLOG, KILL	invoker	invoker	
c_ls: privileged *ls* command; lists contents of a directory	SSA	MAC_EXEMPT, DAC_EXEMPT, AUDITLOG	invoker	invoker	
c_table: shows security tables	SSA	none	invoker	invoker	
c_zaptable: editor for the security tables	SSA	AUDITLOG, CHOWN, DAC_EXEMPT, SETFLBL, MAC_EXEMPT	*bin*	invoker	
disable: stops a printer or a terminal	SO	MAC_EXEMPT, KILL, SETUID, AUDITLOG	*bin*	invoker	
enable: starts a printer	SO	MAC_EXEMPT, KILL, SETUID, AUDITLOG	*bin*	invoker	This *execs* *disable* to disable a printer.
haltsys: closes file systems and halts system	any user	AUDITLOG	invoker	invoker	*haltsys* is directly invocable only by the TSP. It is indirectly invoked via the *halt* option of the privileged trusted shell or the *c_halt* command.
lpmove: moves print requests among queues	SO	MAC_EXEMPT, AUDITLOG	*lp*	*so*	
lpsched: schedules line printer requests from the spool file	SO	MAC_EXEMPT, KILL, AUDITLOG, SETUID, DAC_EXEMPT, SETPRLBL	*lp*	*so*	This is essentially a "permanent" trusted process: the purpose of the SO's *lpsched* command is to start the *lpsched* daemon process.

Trusted Shell TPs (cont.)					
TP Name and Function	Run by	Privileges	uid	gid	Notes
lpshut: stops print daemon	SO	MAC_EXEMPT, AUDITLOG	*lp*	*so*	
mkwtmp: enables session accounting	AA	DAC_EXEMPT, MAC_EXEMPT, SETFLBL, CHMOD, CHOWN, AUDITLOG	invoker	invoker	
mount: mounts a filesystem	any user	MOUNT, AUDITLOG, DAC_EXEMPT, MAC_EXEMPT, CHOWN, WILDCARD, SETPRLBL	*root*	invoker	
passwd: changes log in password	any user	MAC_EXEMPT, AU-DITLOG, SET-FLBL, SIGNAL	*bin*	invoker	
pwd: displays current working directory	any user	none	invoker	invoker	
reject: disallows printer requests	SO	MAC_EXEMPT, AUDITLOG	*lp*	*so*	
rmwtmp: disables session accounting	AA	MAC_EXEMPT, DAC_EXEMPT, AUDITLOG	invoker	invoker	
scheck: verifies security consistency of system	SSA	MAC_EXEMPT, DAC_EXEMPT, ACL	invoker	invoker	Performs security checks (consistency) on system security profiles and all command and data files required for secure system operation. Is also invoked by *init* on system initialization.
star: Secure Tape Archiving Program	any user	MAC_EXEMPT, DAC_EXEMPT, ACL, CHMOD, CHOWN, SET-FLBL, AUDITLOG	invoker	invoker	Both the SO and TSP can copy any file. Other users can copy only their own files (see section 2.8.7, page 68).
stsort: sorts temporary files	any user	none	invoker	invoker	*stsort* is invoked by *star*: hence, it runs in the trusted shell.
treecheck: checks entire file system for security violations	SSA	ACL, DAC_EXEMPT, MAC_EXEMPT	invoker	invoker	Invocable through scheck.
umount: unmounts a filesystem	any user	MOUNT, AUDITLOG, MAC_EXEMPT, DAC_EXEMPT	*root*	invoker	

Permanent TPs					
Name, Ftn.	Initiated by	Privileges	uid	gid	Notes
audswap: monitors available space in audit file	*auditsh*	AUDITLOG, DAC_EXEMPT, MAC_EXEMPT, SETUID	*audit*	*audit*	
cron: executes commands (specified in crontab files) at specified dates and times	/etc/rc	none	*cron*	*bin*	Invoked during system initialization, from */etc/rc*.
getty: polls terminal characteristics, setup	*init* TP	MAC_EXEMPT, DAC_EXEMPT, AUDITLOG	invoker	invoker	
init: initializes terminal, spawns *login* processes	system at boot time	all privileges	invoker	invoker	
nwprt: outputs page headers/footers and banner pages	*lpsched*	none	invoker	invoker	
swapper: swaps processes within the kernel	kernel	kernel process with full access to kernel address space	*root*	*root*	This TP is unique. It runs entirely within the kernel and has no ring three address space. It never makes system calls or performs other privileged operations, but instead calls the swapping primitives directly. It is created during initialization as process zero.
syncclock: synchronizes software and hardware clocks	*cron*	STIME	invoker	invoker	
update: periodically flushes system buffers	/etc/rc	none	*update*	*update*	Invoked during system initialization, from */etc/rc*.

2.7 TCB Protected Resources

Trusted XENIX supports a rich set of objects. The objects supported are ordinary files, directories, System V semaphores, XENIX semaphores, special files, named pipes, unnamed pipes, System V message queues, XENIX shared data segments, System V shared memory segments, and processes. Processes are the only subjects in Trusted XENIX.

2.7.1 Subjects

In Trusted XENIX as in other UNIX systems, the existence of a process is implied by the existence of a non-null `proc` structure entry in the process (`proc`) table. The `proc` structure is permanently resident in main memory and is never swapped out. The `proc` structure contains process attributes including the security label of the process, its identity (uid and gid, described below), its parent process ID, and its status. Finally, each process has an address space. The address space includes a user portion, which is initialized from the process' program file and may be replaced or changed in size by the *exec* and *sbrk* system calls, and may be directly accessed by ordinary user-mode programs. The address space also includes a kernel portion, which is fixed in size, and directly accessible only from within the kernel. Part of the kernel portion is shared by all other processes (kernel code and system-wide kernel data). The other part is accessible only to the owning process, and contains process context (the `u_area`), the definition of the process' address space, and the process' kernel stack. Process attributes stored within the `u_area` include a record of system resources held by the process: open files, shared segments in use, semaphores, etc.

The term "uid" refers to the numeric value corresponding to a "User Name" specified by a user at log in time. An analogous relationship is defined between "gid" and "Group Name." Users can be members of a number of groups, although operate in only one group at a time. The system's internal view is always that of uid and gid. The correspondences are stored in the files */etc/passwd* (for uid) and */etc/group* (for gid).

Each process has a real uid and effective uid associated with it as well as real and effective gids. These IDs are stored in the process' `u_area` and `proc` table entries. Since the effective uid can be different in these two locations, the effective uid in the `u_area` is referred to as the effective uid, and the effective uid in the `proc` table entry is referred to as the saved uid.

The real uid of nonprivileged processes is always derived from the User ID of the user currently executing the process. Only a process with SETUID privilege can change its real uid. The effective uid can be changed with the *setuid* system call, although only a SETUID privileged process can change its effective uid to something other than its real uid (obtained from the *getuid* system call) or saved uid (obtained from the *geteuid* system call). A process which executes a program with the setuid permission bit set has its effective uid set to the owner ID of the program file. This can be reset to the saved uid via *setuid*. The uid used in Discretionary Access Control checks is the effective uid.

Each process has three GPM vectors associated with it: the current, the previous, and the effective. The current GPM vector contains all the privileges that the process is entitled to acquire (i.e., those privileges a program is "installed" with). The effective GPM vector is the GPM vector containing the privileges currently set for the process. The previous GPM vector of a process is a copy of the effective GPM vector of the invoker (another process) of the process at the time of invocation. The current and effective GPM vectors are the same at process creation and diverge as the use of privilege bracketing (see section 2.8.3, page 61) alters the effective GPM vector. The GPM vectors associated with a process are located within the process' `u_area` (accessible only to the kernel).

In addition, processes have the following capabilities: a process can create another process (*fork*), overlay itself with an executable file (*exec*), and send a signal (*kill*) to destroy another process (assuming other constraints on this capability allow it, i.e., MAC and the special constraints on signaling). A process may acquire and possess system resources of various kinds such as memory and CPU time.

Process Creation and Modification

When invoked, the *fork* system call creates a process (known as the child) that is the exact duplicate of the "parent" (invoker of *fork*) except for the following:

- The return values to the parent and child are different. The value of the child's process ID is returned in the parent process and the value zero is returned in the child process.
- The child process has a unique process ID.
- The child process has a different parent process ID.
- The child process has its own copy of the parent's file descriptors. Each descriptor shares a common file pointer with the corresponding file descriptor of the parent.
- The child process' resource usage values (utime, stime, cutime, and cstime) are set to zero.
- The child process' `semadj` values (used to restore semaphore value after abnormal process termination) values are cleared.

Trusted XENIX will allocate primary or swap space memory for the new process and make an entry for the child process in the `proc` table. The *brkctl* system call is used to allocate or deallocate memory segments, and the *brk* system call is used to change the size of data segments. The *lock* system call locks a process in memory (text and data segments). Processes or process groups can be manipulated with the *proctl* system call as follows. This system call will allow a process that is greater than the swapper size to execute, will allocate contiguous memory and then deallocate it. Without PROCTL privilege, processes can invoke this system call to affect only those processes whose real or effective uid is equal to the real or effective uid of the invoker (except for process 0 and process 1), and whose security levels are greater than or equal to the security level of the invoker.

The *exec* system call transforms the calling process by overlaying it with the executable file (known as the "new process file") specified in the *exec* call. The *namei* routine, which does pathname resolution, is called to get the inode of the executable file for DAC and MAC permission checks, and the file table is searched to check that a file open for writing is not being *exec*ed. When a process issues an *exec*, a new stack is allocated and overlaid on the old stack and a new executable file is overlaid on the user address space. File descriptors open in the calling process remain open in the transformed process, except for those whose close-on-exec flag is set. If the setuid mode bit of the new process file is set, *exec* sets the effective uid of the transformed process to the owner ID of the new process file. Similarly, if the setgid mode bit of the new process file is set, the effective gid of the transformed process is set to the gid of the new process file. The real uid and real gid of the transformed process remain the same as those of the calling process. The transformed process also retains the following attributes from the calling process: pid, parent pid, `semadj` values, signal values (e.g., whether to ignore or terminate[22]), and process security level. The effective and maximum privilege

[22] However signals set to be caught by the calling process are set to terminate the transformed process.

vectors are both computed as the bitwise "or" of the effective privilege vector of the calling process and the privilege vector of the *exec*ed file. The transformed process cannot be traced by any process with less privilege. Various forms of *exec* (e.g., *execl*, *execv*, *execlp*, *execvp*, *execve*) are available to a programmer. The basic difference lies in parameter specification; all are implemented as calls to *execve*, which is the actual kernel routine.

All processes, except for process 0, are created only through the execution of a *fork* system call by some other process. Process 0 is a special process (created by the boot routines) which, in turn, creates a child process known as process 1 or the *init* process. Process 0 then becomes the kernel's *swapper* routine, while the *init* process becomes the ancestor of every other process in the system. *init* clears all console display memory between console log ins via *ioctl*. The device drivers also clear all their physical memory areas. *init* also does a *chmod* 600 for the terminal so that other processes cannot write to a login prompt screen. For each terminal in the system, the *init* process forks a copy of itself which in turn will invoke the *getty* TP (which will acquire the user's terminal characteristics and save them in the */etc/utmp* file which contains one entry for each terminal in the system) and the *login* TP to perform user log in.

User Log in

The user is logged in by the TP *login*, which first invokes the password routine which authenticates the user. Immediately following successful identification and authentication, *login* checks to ensure that the User Maximum Level (UML) dominates the Terminal Minimum Level (TMinL). *login* then asks the user to specify the desired group and security access level. *login* ensures that the user is a member of the specified group and that the Group Maximum Level (GML) dominates the TMinL. If the user does not specify the group and/or security level then the appropriate default values are used for the item not specified (previously set by the SSA in */etc/security/s_user*). The security level entered by the user (or the default as appropriate) is compared to the UML, the GML, and the Terminal Maximum Level (TMaxL). If the requested level exceeds any one of these levels or is not greater than the TMinL, then an error message is printed and the prompt reissued. Upon successful completion of these checks, *login* will set the Current Process Level (CPL) to the requested security level, or the default security level if none was specified. *login* will make the necessary entries into the */etc/utmp* file to show that the terminal is now connected to the user process and set the terminal level to the CPL by modifying the inode of the terminal's special file. Finally, *login* *exec*s the user's shell to set up the user's initial environment.

Process Termination

When the user process is terminated (i.e., logout) the process manager notifies the *init* process (process 1) of the death of its descendant at which time *init* forks another copy of itself (which, in turn, invokes *getty*) to the terminal to await the next user.

When a process is terminated (via *exit* or receipt of signal) the following actions occur. All open file descriptors are closed, each shared memory segment is detached, outstanding semaphore operations left by the terminating process are closed, and all locks (process, text or data) are unlocked. If the terminating process is a process group leader, the SIGHUP signal is sent to all processes in the process group. *init* becomes the parent of the terminating process' children. If the parent of the terminating process is waiting on an event, it is notified of the termination. If the parent is not waiting, the terminating process is turned into a "zombie" process (occupies a slot in the process table but has no user or kernel space available to it). Zombie processes are removed from the process table when their parents exit.

2.7.2 Objects

The objects in Trusted XENIX can be divided into two broad classes: those objects with a file system representation and those without. Objects with a file system representation share a common access decision mechanism and use the same data structures and code to implement access control. Access is granted based on the permission bits, ACLs (if present), and security label stored in the inode. When the inode is brought into main memory (into the inode table), extra information about the dynamic access attributes is included in the in-core inode copy. For those objects without a file system representation, a special TCB data structure with its own access control information (e.g., permission bits and ACLs) is used. The objects of Trusted XENIX are listed in table 2.4 to show which are file system objects and which are not.

Trusted XENIX Objects	
File System	Non File System
Ordinary Files	System V Semaphores
Directories	System V Message Queues
XENIX Semaphores	System V Shared Memory Segments
Special Files	Processes
Named Pipes	
XENIX Shared Data Segments	
Unnamed Pipes	

Table 2.4. Trusted XENIX Objects

Object Descriptions

The access control information for file system objects is stored in the object's inode. In addition to complying with the security policy enforced for the object type, a process must also have search access to each directory named in the path to the object. The access control information for non-file system objects is stored in special TCB data structures specific to the object type.

Each of the following subsections briefly describes the object type, modes of access allowed to that object type, and the conditions necessary to exercise each access mode.

Each of the object type descriptions includes a brief discussion of the conditions necessary to exercise each mode of access defined for the object type. The notation used for these discussions is as follows: "SL(process)" represents the mandatory security level of the subject, "SL(<object type>)" represents the mandatory security level of the object specified, "$\geq$" represents "dominates", "$\leq$" represents "is dominated by", and "=" represents "equals" (for a more detailed description of the "dominates" relation see page 59). In each of the following descriptions the "effective discretionary access mode" refers to the result of the discretionary access control check as described in section 2.8.1 on page 56. Access modes not specified for a particular object type have no effect on the access calculation.

Ordinary Files (table 2.5): Files are the primary information containers for the system. They are represented by an inode. Essentially a file is a device inode-number pair. The name of the file and the inode are both stored in a directory, forming an entry for that file. Each file consists of zero or more disk blocks, which are either direct (that this, they contain data) or indirect (they point to disk blocks which may contain data, or more pointers).

Access Type	Comment
read (r)	A process may exercise read access if and only if: 1. the effective discretionary access mode includes “r” and 2. SL(process) $\geq$ SL(file) A process granted read access can execute system calls that cause data to be fetched (read) from the file.
write (w)	A process may exercise write access if and only if: 1. the effective discretionary access mode includes “w” and 2. SL(process) $=$ SL(file) A process granted write access can execute system calls that cause existing data in the file to be modified or new data be appended to the file.
execute (x)	A process may exercise execute access if and only if: 1. the effective discretionary access mode includes “x” and 2. SL(process) $\geq$ SL(file) A process granted execute access can cause the program to be loaded and executed or the shell script to be forked and executed.
null (-)	A process cannot access the file in any way.

Table 2.5. File Access Rules

Directories (table 2.6): Directories are the filesystem data structure responsible for providing the tree structure for the filesystem. Directories contain a mapping of file name component to inode number. Each directory is responsible for providing the inode number of each object immediately subordinate to it. By consulting this mapping at each step along the path to a file, the kernel is able to translate full pathnames into the proper inode number of the desired object.

Directory entries are known as links. Each link provides a name-to-inode number mapping for only one object; however, a single filesystem object may have many links that "point" to it, thus allowing more than one pathname to describe an object.

For most file system objects, the mandatory access compatibility rule[23] requires that the security level of the object is always equal to the security level of the containing directory. Directories are the primary exception to this rule in that the security level of a directory need only dominate the security level of the containing directory. An "upgraded directory" is a directory with a security level that is higher than the security level of the containing directory. The TCB ensures that a directory's label dominates the label of its parent directory. Thus, as one traverses a path from the root to an object, the labels are monotonically non-decreasing.

System V Semaphores (table 2.7): System V semaphores are objects that are used to implement a process synchronization mechanism. System V semaphores can take any integer value. System V semaphores are not part of the file system, but rather are managed completely as an internal TCB data structure. This internal data structure has a component much like an inode that contains the access control information associated with the semaphore. Semaphore control structures and data are stored within the kernel's data segment. The kernel maintains a fixed size table for semaphore control information.

System V semaphores are actually semaphore sets which can accept a list of semaphore operations. Each set of operations is performed atomically; that is, the entire operation set is performed together. If one operation of the set causes the calling process to sleep, then all previous operations in the set are undone and restarted from the beginning when the process reawakens. Semaphore sets are created via the *semget* system call. *semget* returns a set id which is used to identify the set for semaphore operations. Semaphore sets that are to be long-lived and publically known can be created using a user specified key value. The number of semaphores in the set is specified when the set is created.

Synchronization is accomplished via the *semop* system call. The inputs to *semop* are the ID of the semaphore set to use, a list of semaphore operations, and a count of the number of operations in the list. Each operation specifies which semaphore in the set to operate on, the operation to be performed, and flags to control how the operation is performed.

XENIX Semaphores (table 2.8): XENIX semaphores provide a mechanism similar to the System V semaphores. XENIX semaphores differ from System V semaphores in that XENIX semaphores can take values of zero or one only, and XENIX semaphores are part of the file system. XENIX semaphores are implemented directly in their corresponding inodes. The inode, marked internally as a semaphore, points to a zero length file.

A XENIX semaphore is created via the *creatsem* system call using the file system name space to name the semaphore. Any process wishing to synchronize using this semaphore accesses it via the *opensem* system call. Both *creatsem* and *opensem* allocate file descriptors for semaphores. This system call returns a semaphore id which is then supplied to the synchronization calls, *sigsem* and *waitsem*. The synchronization calls are both

[23]The "compatibility rule" is described in detail on page 25 of *Secure Computer Systems: Unified Exposition and Multics Interpretation* [BELL].

Access Type	Comment
read (r)	A process may exercise read access if and only if: 1. the effective discretionary access mode includes "r" and 2. $SL(process) \geq SL(directory)$ A process granted read access to a directory can ascertain the name of any entry in the directory.
modifyentry (w)	A process may exercise modifyentry access if and only if: 1. the effective discretionary access mode includes "w" and 2. $SL(process) = SL(directory)$ A process granted modifyentry access to a directory can cause current entries to be unlinked (removed) or new entries to be linked (added).
search (x)	A process may exercise search access if and only if: 1. the effective discretionary access mode includes "x" and 2. $SL(process) \geq SL(directory)$ A process granted search access to a directory can use that directory's name in a pathname. In addition, the attributes of file system objects (e.g., date and time of last change, protection characteristics, etc.) in that directory are available to a process granted search access to the directory provided $SL(process) \geq SL(file)$.
null (-)	A process granted null access to a directory cannot access the directory in any way.

Table 2.6. Directory Access Rule

Access Type	Comment
observe (r)	A process may exercise observe access if and only if: 1. the effective discretionary access mode includes "r" and 2. SL(process) = SL(semaphore) A process granted observe access to a System V semaphore may examine the value of the System V semaphore.
read/write (rw)	A process may exercise read/write access if and only if: 1. the effective discretionary access mode includes both "r" and "w" modes and 2. SL(process) = SL(semaphore) A process granted read/write access to a System V semaphore may examine and/or alter the value of the System V semaphore.
null (-)	A process granted null access to a semaphore cannot access the semaphore in any way.

Table 2.7. System V Semaphore Access Rules

Access Type	Comment
read/write (r)	A process may exercise read/write access if and only if: 1. the effective discretionary access mode includes "r" and 2. SL(process) = SL(semaphore) A process granted read/write access to a XENIX semaphore may examine and/or alter the value of the XENIX semaphore.
null (-)	A process granted null access to a XENIX semaphore cannot access it in any way.

Table 2.8. XENIX Semaphore Access Rules

atomic operations; an operation on a given semaphore cannot begin until any other operation in progress completes.

Special Files (table 2.9): Special files are used to represent devices and can be manipulated by user processes in exactly the same manner as files. There are two general classes of I/O devices: block devices and character devices. Block devices correspond to secondary storage devices such as hard disks, diskettes, and tapes. These devices must have fixed size blocks which are directly addressable. Non-block devices (display, printer, keyboard) are implemented as character devices. Device drivers are represented by entries in the block or character switch array (`bdevsw` or `cdevsw`), and define operations for the special files (e.g., read, write). Operations on these two types of special files translate into physical operations on the corresponding devices.

For special files that represent single-level devices, the level of the special file as recorded by the file system is used. For special files that represent multi-level devices, discretionary access to the special file is restricted to the trusted process that is responsible for manipulating the device and controlling its sensitivity level. The trusted process is then responsible for all mandatory access control on the device and manages the labels in a manner best suited for the device in question.

In addition, special files have a defined maximum security level and minimum security level. The maximum security level must dominate the minimum security level. No device is ever allowed to operate outside the range of security levels specified by its maximum and minimum security levels.

The only exceptions to this are the special devices */dev/tty* and */dev/null*. The pseudo terminal device (*/dev/tty*) is used by processes executing the open system call without knowing the real name. When a process attempts to open */dev/tty*, the kernel redirects the open to the user's actual terminal. In order for the initial open to succeed, */dev/tty* has the WILDCARD label. The null file, */dev/null*, also has the wildcard label. This is acceptable since writes to it are discarded, and reads from it always return an end-of-file.

Named Pipes (table 2.10): Named pipes are used as communication buffers between two processes. Named pipes provide an inter-process communication facility that manages messages in a first-in, first-out manner. Their implementation allows processes to communicate even though they do not know what processes are on the other end of the pipe. Pipes are implemented using the filesystem for data storage. The kernel assigns an inode and a directory entry for named pipes, as for other files.

Access Type	Comment
read (r)	A process may exercise read access if and only if: 1. the effective discretionary access mode includes "r" and 2. SL(process) = SL(special file) A process granted read access to a special file may open the file for read.
write (w)	A process may exercise write access if and only if: 1. the effective discretionary access mode includes "w" and 2. SL(process) = SL(special file) A process granted write access may open the file for write.
null (-)	A process granted null access to a special file may not access the device represented by that special file in any way.

Table 2.9. Special File Access Rules

Processes use the *mknod* system call to create named pipes and the *open* system call to open named pipes. Once open, processes use the regular system calls for files (e.g., read, write and close) when manipulating named pipes.

Unnamed Pipes: Unnamed pipes provide an inter-process communication facility like named pipes. Unnamed pipes do not have names in the file system; however, they do use the file system data structures (e.g., inode). Unnamed pipes are used as communication buffers between a child process and its parent process.

Unnamed pipes, unlike named pipes, are transient; when all processes finish using the pipe, the system reclaims its inode. Processes use the *pipe* system call to create an unnamed pipe. The system call returns two opened file descriptors, one opened for reading and one opened for writing (unlike named pipes which do not do the opening as part of creation), and creates the corresponding file table entries for the pipe. I/O to unnamed pipes is always synchronous. If the pipe is full of data, a write request will block the requesting process until another process reads sufficient data from the pipe. A read from an empty pipe blocks the requesting process.

Since unnamed pipes can be shared only between a child process and its parent process and only if passed on to the child from the parent as part of the process context duplicated when the child is *fork*ed, no other access control is enforced.

System V Message Queues (table 2.11): System V message queues are containers for messages. They allow processes to exchange data in units of messages, collections of data of a specific size, rather than the unstructured stream of data provided by pipes. System V message queues are primarily used to hold requests to server processes. Each message in a queue has an attribute named `type`. This allows the receiver to choose the messages it receives (or order of messages to receive, etc). Usually the `type` will signify whether the

Access Type	Comment
read (r)	A process may exercise read access if and only if: 1. the effective discretionary access mode includes "r" and 2. SL(process) = SL(named pipe) A process granted read access to a named pipe may extract the oldest information in the named pipe. Extracting the information deletes it from the named pipe.
write (w)	A process may exercise write access if and only if: 1. the effective discretionary access mode includes "w" and 2. SL(process) = SL(named pipe) A process granted write access to a named pipe can only append information to the named pipe.
null (-)	A process with null access to a named pipe cannot access the named pipe in any way.

Table 2.10. Named Pipe Access Rules

message is from the client to the server or from the server to the client (two channels would be necessary for this bi-directional communication using pipes).

Message queues are created via the *msgget* system call, which takes a user specified key value and returns a queue id to be used for later queue operations. Messages are sent to a queue via *msgsnd* and retrieved via *msgrcv*. Unlike pipes, processes can write to a message queue even if there is no process waiting for message entry on the queue. Processes may request message queue operations to be synchronous or asynchronous (i.e., can block until the request can be satisfied or can be notified that the operation failed).

System V message queues have no file system representation and are completely managed by memory resident tables available to the TCB. Message queue control structures and message data are stored within the kernel's data segment. The fixed sized tables used within the kernel do, however, mimic the file system DAC policy by providing a capability to specify protection bits or an ACL for the System V message queue. The protection bits or the pointer to the ACL is kept in the table. Part of the memory-resident table is the label for the System V message queue. That label is used for all MAC decisions concerning the message queue.

XENIX and System V Shared Memory Segments

Two shared memory mechanisms are provided: XENIX Shared Data and UNIX System V Shared Memory. Both mechanisms support interprocess communication via segments of common memory that can be mapped into the address spaces of multiple processes. The key advantage here is the reduction in overhead of multiple copies normally necessary to handle asynchronous reading and writing of the file by different processes, which relieves the kernel from responsibility for managing data structures and acting as intermediary between process and data (as is the case for pipes and message queues).

Access Type	Comment
read (r)	A process may exercise read access if and only if: 1. the effective discretionary access mode includes "r" and 2. SL(process) = SL(message queue) A process granted read access to a System V message queue may extract the oldest message from the System V message queue. Extracting the message deletes it.
write (w)	A process may exercise write access if and only if: 1. the effective discretionary access mode includes "w" and 2. SL(process) = SL(message queue) A process granted write access to a System V message queue may only add messages to the System V message queue.
null (-)	A process granted null access to a System V message queue cannot access the System V message queue in any way.

Table 2.11. System V Message Queue Access Rules

Access Type	Comment
read (r)	A process may exercise read access if and only if: 1. the effective discretionary access mode includes "r" and 2. SL(process) = SL(shared segment) A process granted read access to a shared segment may examine the shared segment.
write (w)	A process may exercise write access if and only if: 1. the effective discretionary access mode includes "w" and 2. SL(process) = SL(shared segment) A process granted write access to a shared segment may alter the contents of the shared segment by adding new data to the segment or by altering the existing data.
null (-)	A process granted null access to a shared segment cannot access the shared segment in any way.

Table 2.12. XENIX Shared Data Segment Access Rules

The actual shared segment is reached by inodes for both types (as explained below) of segment. The segment's data structure includes a permission structure, size, usage information, etc., and is also pointed to by the TCB's shared data table. The process table maintains a list of shared memory control blocks for each process with shared segments attached. Each control block contains the inode pointer, start address, dynamic permission flags, and a pointer to the next control block for the associated process.

XENIX Shared Data Segments (table 2.12): The *sdget* system call creates a XENIX shared data segment, or attaches an existing segment to the calling process. The segment is identified by a path name in the file name space. The *sdfree* system call removes a XENIX shared data segment from the calling process' address space. When the last process detaches from a segment, the segment is destroyed.

XENIX Shared Data Segments also support system calls (*sdgetv* and *sdwaitv*) to synchronize cooperating processes using shared data segments. Therefore the security policy mandates read- and write-same.

System V Shared Memory Segments (table 2.13): The system calls for System V shared memory have the same form as those for message queues and System V semaphores, where user supplied keys are specified, rather than the XENIX Shared Data Segment interface which uses filesystem supplied inodes. *shmget* creates System V shared memory segments with segment ids or user specified key values. The system calls *shmat* and *shmdt* request that a particular System V shared memory segment be mapped into or removed from the address space of the requesting process (read-only or write-only mode can be specified).

Unlike XENIX shared memory segments, System V shared memory segments do not have a user-visible pathname but rather are referenced via a shared memory segment identifier which the TCB maps to a file. So although the Shared Memory Segments are not named from the filesystem, the filesystem is used

Access Type	Comment
read (r)	A process may exercise read access if and only if: 1. the effective discretionary access mode includes "r" and 2. SL(process) $\geq$ SL(shared segment) A process granted read access to a shared segment may examine the shared segment.
write (w)	A process may exercise write access if and only if: 1. the effective discretionary access mode includes "w" and 2. SL(process) = SL(shared segment) A process granted write access to a shared segment may alter the contents of the shared segment by adding new data to the segment or by altering the existing data.
null (-)	A process granted null access to a shared segment cannot access the shared segment in any way.

Table 2.13. System V Shared Memory Segment Access Rules

for storage. The TCB maintains a System V Shared Memory Segment Table which provides the mapping between user supplied key and the inode (of type System V Shared Memory) which the TCB uses to get to the actual shared memory segment. The System V Shared Memory Segment Table entries include the key, status information and a pointer to an inode. That inode is of type System V Shared Memory and as for System V Semaphores the TCB interprets the union structure for the shared segment. The inode contains (in that structure) the starting memory address of the segment, length, flags, in-core reference count, creator information, etc. From this point the TCB does the same thing for both types of shared segment.

Processes: Although processes are the subjects in the Trusted XENIX system, when used as the recipient of an inter-process signal they must be treated as objects. Processes are represented by entries in the `proc` table.

Processes may send signals to other processes if and only if:

1. The sending process' effective uid or real uid is the same as the receiving process' effective uid or real uid,
2. SL(receiving process) $\geq$ SL(sending process), and
3. The receiving process' current privilege vector (see section 2.8.3, page 60) is a subset of the sending process' current privilege vector.

2.8 TCB Protection Features

This section describes the various features that are used to protect the resources from unauthorized access by the subjects on the system. The features to be discussed include DAC, MAC, the use of privilege on the system, auditing, Object Reuse, the trusted path implementation, archiving, printer spooler management, and the system deflection mechanism.

2.8.1 Discretionary Access Controls

DAC allows subjects to grant or deny access to objects at their own discretion. All objects (except processes) are identified by the owner's effective uid and effective gid at the time of creation (see section 2.7.1 on page 42).

Trusted XENIX provides three DAC mechanisms: protection bits, access control lists, and setuid/setgid bits. ACLs and protection bits can be applied to all Trusted XENIX objects except processes and unnamed pipes. Three access types are available in Trusted XENIX: read (r); write (w); and execute (search access for directory objects) (x).

Protection Bits

Stored in each file's inode are 11 security-relevant bits associated with that file; the first two bits are the setuid and setgid bits and the remaining nine bits specify read, write, and execute access for the object's owner, its group, and everyone else. Non-file objects have only the nine security-relevant bits, and these are stored in kernel tables. The first set of three bits (rwx) represents access for the owner, the second for the owning group, and the third for everyone else. When a user attempts to access an object, the system first checks to determine whether the user is the "owner;" if so, the authorized access is granted (even if the access is "no access"), and the search stops. If the user is not the owner, the system next checks to see whether the user is a member of the group to whom the file belongs; if so, the access authorized to that group is granted (even if the access is "no access"), and the search stops. Finally, if the user is neither the owner nor a member of the group to which the file belongs, the system checks the accesses given to everyone else, and that access is granted.

Group membership for users is set by the System Administrator along with a default log in group. Users can change their current group only by logging out and logging back in. Each file is represented in the system by an inode, which holds information about the file, such as its physical location on the disk, its size, its owner and group, and the permission bits associated with it. The owner's effective uid and the effective gid are recorded in the inode at file creation. The *chown* and *chgrp* commands allow the owner of a file to change its owner or group; however, doing so will clear any setuid or setgid bits set on the file. The only other users who can change a file's ownership or group are the TSP, using the *chown* command, and the SSA, using the *c_chowner* TP from the privileged trusted shell. The *chmod* command allows a user to change the protection bits on a file using a triple of letter or octal-digit representations of the permissions for owner, group, and everyone else.

The *umask* command specifies the default protection bit settings when a file is created. Any bits set to "1" in the `umask` will be cleared upon file creation. The default `umask` for the system (when the user has not specified a `umask`) is 077, meaning that only the user has access to the object. For files automatically created by the system (as opposed to explicitly by a user), such as output redirection and file backup by an editor, the `umask` is masked onto the shell's default protection of 0666. Thus, the protection applied to files with the default `umask` of 077 is 0600 (read/write for the user only):

Default **umask**	000111111
Shell default protection	000110110110
Result (0600)	000110000000

Access Control Lists

The second DAC mechanism provided by Trusted XENIX is the ACL, which can be associated with files, directories, System V message queues, System V semaphores, System V shared memory, XENIX semaphores, XENIX shared memory, named pipes, and devices. ACLs provide a more convenient way of specifying read, write, and execute access control for individual users, groups, and everyone else. ACLs are represented as files, and the number of entries in an ACL is limited only by the maximum size of a file. The inode number for the ACL file and the name of the ACL file are stored in the inode of the file whose DAC information the ACL holds. ACLs have the same security label as the file they protect. ACLs are stored in the `root`-owned directory */.ACL* with protection 700 (rwx for `root` only); each file system has its own */.ACL* directory. Users cannot access ACL files directly; only the TCB can access ACLs, and only the owner of the file protected by an ACL can modify it. Externally, ACLs are of the form: "user-id:group-id:access-type". The user-id and group-id are the names of the user or group to which access is being specified, respectively, and can be replaced by the wildcard keyword "ALL", indicating that the specified access applies to all users or all members of the group. If the user-id or group-id is omitted, that field is implicitly set to "ALL". The access-type field may be any alphabetical combination of the letters r (read), w (write), and x (execute); a single octal digit corresponding to the set of three permissions; or blank, specifying no access.

ACLs are not created automatically, and no user-settable defaults exist. As part of the high-level interface (untrusted user commands, interpreted by the shell), the owner of the object can use the *acl mk* command to create an ACL that duplicates the object's current protection bit specification. Then the owner can use *acl add* and *acl rm* to add and delete specific entries. Commands to list and purge an ACL are also provided. When an ACL is created, the low-order nine protection bits are set to zero (no access). The high-level interface sorts the entries from most specific user-id to least specific as they are added to the list.

SETUID Protection

The two high-order protection bits of a file are the setuid and the setgid bits. These bits may be turned on only if the file is executable. The file owner can set these bits only by using the privileged *chmod* command available only through the trusted path mechanism. When a file with one of the bits turned on is executed, the resulting process takes on the effective identity of the file's owner or group and assumes all of the discretionary access rights associated with that identity.

This setuid/setgid mechanism is a very useful, and potentially dangerous, feature if left uncontrolled. Two of the most common abuses in UNIX systems historically have been the overwriting of setuid files and the actions of Trojan horses. In the first case, a malicious user locates a setuid/setgid file to which he or she also has write access, overwrites it with a copy of the shell, and then proceeds to operate with all of the discretionary privileges of its owner/group. In the second case, a malicious user creates a Trojan Horse program which, when executed, creates a setuid/setgid file with the victim as owner/group; so that whenever the malicious user executes it, he or she assumes the identity of the victim/victim's group.

Trusted XENIX provides protection against these two abuses of the setuid/setgid mechanism. If a setuid/setgid file is opened for write, the setuid/setgid bit is cleared, thus eliminating the first avenue of attack. The confinement of the capability to set the setuid and setgid bits to within the trusted shell via the privileged option of the *chmod* command eliminates the possibility of a Trojan horse's setting the setuid or setgid bit on a file.

Programmers are still responsible for creating "well-behaved" setuid/setgid programs, meaning that a user running the setuid/setgid program should not be able to escape to a sub-shell without resetting the effective uid or the effective gid. Otherwise, the user would be able to access all files of the program's owner/group.

DAC Algorithm

When a user process attempts to access an object, the DAC mechanism mediates the access using the following algorithm.

1. Check to see whether setuid bit is set. If so, grant DAC privileges of the file's owner.
2. Check to see whether setgid bit is set. If so, grant DAC privileges of the file's group.
3. Check to see whether an ACL exists for the object. If so, scan the entries sequentially until an entry matching either the user's effective uid or effective gid is found. If a match is found, grant that access; else grant no access.
4. If no ACL exists, check the nine low-order protection bits. Scan the permission bits from left to right, and grant the user the permissions specified for the first match found.

As mentioned above, the high-level interface sorts the ACL entries from most specific to least specific. If for some reason the user chooses to write a routine to add entries to the ACL list using the system calls, rather than using the delivered interface, and if the user's routine does not include a sort function, the entries would be unsorted, as the system call merely appends an entry to the end of the file. In such case, the system cannot guarantee that the most specific access rule is applied, since the first match will be selected.

2.8.2 Mandatory Access Controls

The mandatory security policy restricts the type of operations a subject may perform on an object based on the relationship between the subject's security level and the object's security level. For each subject/object type supported by Trusted XENIX, a set of mandatory policy rules have been defined which control creating subjects and objects; deleting subjects and objects; and reading, writing, and executing objects. These rules are patterned after the security model defined by Bell and La Padula [BELL], although some of the Trusted XENIX rules are more restrictive than their Bell and La Padula counterparts [INTP].

Each subject and object in Trusted XENIX is assigned a sensitivity label, which comprises one hierarchical security or clearance level and a set of non-hierarchical categories; this sensitivity label is used as the basis for all MAC decisions. The sensitivity label is internally represented as a nine-byte vector stored as the

Object Type	Rule Enforced
File Special File (Device) Directory Shared Memory Segment ACL	Read/Execute, Search (Directory) *iff* $L_S \geq L_O$ Write *iff* $L_S = L_O$
Named Pipe Semaphore Message Queue XENIX Shared Data Segment	Read/Write (open/close) *iff* $L_S = L_O$
Process	Signal (kill) *iff* $L_S \leq L_O$
All	Null for all other cases

Table 2.14. Access Rules for Non-Trusted Subjects

`seclab` field of the `inode`. The first byte is an integer representation of the hierarchical level of the subject or object, so that 255 levels plus the WILDCARD (see section 2.8.3 on page 64) may be represented. The remaining eight bytes represent a bit-mapped set of from zero to 64 non-hierarchical categories. The SSA defines these levels and categories.

The security levels of the Trusted XENIX system are partially ordered by the relationships *equals*, *dominates*, *dominated by*, and *isolated from*.

- The security level L1 is said to *equal* that of L2 if the security level of L1 is equal to that of L2, and the category set of L1 is equal to the category set of L2.
- The security level L1 is said to *dominate* that of L2 if the security level of L1 is greater than or equal to that of L2, and the category set of L2 is a subset of that of L1.
- The security level L1 is said to be *dominated by* that of L2 if the security level of L1 is less than or equal to that of L2 and the category set of L1 is a subset of that of L2.
- The security level L1 is *isolated from* the security level L2 if L1's category set is not included in L2's category set, and L2's category set is not included in L1's category set.

The access rules for non-trusted subjects are depicted in table 2.14.

The SSA defines System and User security profiles. The System Profile comprises:

- The System Security Map (i.e., classification and category tables), which includes:
  - the system maximum level (HIGH) and the system minimum level (LOW),
  - the allowed classification numbers and category sets (0 – HIGH for levels and subsets of ALL bitmap category sets), which cannot be deleted, changed, or duplicated,
  - the printed (external) names for classification numbers and category sets;

- The Device Maximum Level (DEVmax) for each attached physical device, where DEVmax $\leq$ HIGH;
- the Device Minimum Level (DEVmin) for each attached physical device, where DEVmin $\geq$ LOW.

The User Profile comprises:

- The User Maximum Level (UML) for each user, where UML $\leq$ HIGH;
- The User Default Level (UDL) for each user, where UDL $\geq$ LOW;
- The Group Maximum Level (GML) for each group, where GML $\leq$ HIGH.

The security tables are verified by the TSP by running the *scheck* TP as the last phase of secure system installation. This check verifies that the database defining the system security profiles satisfies the security requirements for the system, and that the attributes of all command files, data files, and related directories required for secure operation of the system conform to their specifications in the Installation Table. On system power-on or re-boot, *scheck* is run from */etc/rc*, and must run successfully in order for the system to transition to multi-user mode. If *scheck* fails, the system will not come up and the user will be forced to re-boot. The SSA may execute *scheck* from the trusted shell during multi-user operations; if it fails, the TSP must put the system into maintenance mode to take corrective action.

2.8.3 Privilege

A variety of privilege mechanisms are used in Trusted XENIX. These include mechanisms which distribute the power of the traditional UNIX "superuser," trusted processes, a mechanism for checks and balances among privileged roles on the system (the "power hierarchy"), and use of the WILDCARD security label. These mechanisms are discussed below.

The Generalized Privilege Mechanism

The single, omnipotent privilege normally accorded to the superuser (uid 0) during normal multi-user operations in a traditional UNIX system has been decomposed in order to allow for a finer granularity of control via a GPM. The GPM is implemented as a string (vector) of 64 bits of which 36 are used to define distinct privileges, each represented by a single bit.[24] A GPM vector is located in the inode of every file. The default GPM vector is all zeros, indicating no privileges. Each process has three GPM vectors associated with it, each of which are explained in section 2.7.1 on page 42.

In the case of a process created by a *fork* call, the child process inherits the privilege vectors of its parent. In the case of a process transformed as the result of an *exec* call, the Current GPM vector of the resulting (transformed) process is constructed by taking the union of the Effective GPM vector of the calling process and the privilege vector (located in the inode) of the new process file. The Effective GPM vector will initially be identical to the Current GPM vector, and the Previous GPM vector will be set to the invoking process Effective GPM vector at the time of invocation.

[24] The WILDCARD privilege is ineffective by itself and will always require at least one additional bit to be set.

The Current GPM vector initially assigned to the user's process at log in is the GPM vector stored in the inode of the user's login shell (typically a null vector for all non-trusted users). The Effective GPM vector is set to the same value, and the Previous GPM vector is set to null.

Privilege bracketing allows a process to drop privileges from its Effective GPM vector and/or reacquire any privilege that is contained in its Current GPM vector. Use of the bracketing feature has no effect on the Current GPM vector, the Previous GPM vector, or the GPM vector stored in the inode of the file in execution. Bracketing is accomplished through the use of *lapse* and *acquire* system calls and is used by TPs in this system for the bracketing of the MAC_EXEMPT and DAC_EXEMPT privileges. Privileges may also be permanently dropped (i.e., they may not be reacquired using the *acquire* system call) by a process using the *drop* system call.

The GPM vector is accessed by the system calls *getfpriv* or *getppriv*, which return the privileges of the specified pathname, or process respectively. The system call *privilege* may be invoked from within a trusted process to determine the privileges that are currently "on" in the trusted process' Effective GPM vector. The GPM vector located within a file's inode may be set by the TSP using the *setfpriv* system call.

Privileges currently defined for the GPM are shown in table 2.15.

Privileged System Calls

Privileged system calls are those system calls that may be executed only by some subset of the processes on the system. Trusted XENIX confers privilege to execute these calls through use of the Previous GPM vector (i.e., appropriate privilege bit set in the GPM vector of the calling process).

Table 2.16 gives a list of privileged system calls, their functions, and the privilege needed to invoke the system call.

Non-Privileged System Calls with Privileged Options

Non-privileged system calls are those system calls which may be executed by any user. Some non-privileged system calls contain a range of options, of which only a subset is available to the general user population. The remaining options are called privileged options and are available only to processes whose Effective GPM vector has the appropriate privilege bit(s) set.

Table 2.17 lists the non-privileged system calls that possess privileged options in Trusted XENIX.

Power Hierarchy Mechanism

Trusted XENIX uses a "Power Hierarchy Mechanism" to implement a hierarchy among the *TSP*, *SSA*, *SO*, *AUDIT*, and all other user groups. It is utilized only by trusted processes (though not by all) and determines whether a particular user may alter information that either describes or is the property of another user (for example, passwords). The hierarchy is maintained via the association of a group tag with every group, both administrative and user, on the system.

The group tag consists of a number and a group set. Using the number portion of the group tag, all groups can be ordered from highest to lowest power, with some groups being of equal power. The second

Privilege	**Result**
ACCT	run *acct*, enables/disables process accounting
ACL	run *aclcreat*, *aclopen*, *aclrm* with unrestricted access to all users
AUDIT	run *audit*, controls auditing
AUDITLOG	run *auditlog*, write an audit record
CHMOD	run *chmod*, change mode of file (the 11 UNIX file protection bits), with unrestricted access to all users
CHOWN	run *chown*, change owner, with unrestricted access to all users
CHROOT	run *chroot*, change root directory
DAC_EXEMPT	exempt from DAC checks
DEVICE	use an in-use device which was opened in exclusive mode
DUMPKMON	run *dumpkmon*
FORK	run *fork*, allowed to exceed per-user limits
KILL	run *kill* against any process
LINK_DIR	run *link*, *unlink* with a directory as object
MAC_EXEMPT	exempt from MAC checks
MAINT_MODE	indicates process is in maintenance mode
MKNOD	run *mknod* for all types of files
MOUNT	run *mount*, *unmount*
MSGCTL	run *msgctl*, to own all message queues
NICE	run *nice* to increase process priority
PLOCK	run *plock*, locks process, text, or data in memory
PROCTL	run *proctl*, performs various functions on active process or process groups
SEMCTL	run *semctl* to own all semaphores
SETFLBL	run *setflbl* to set file labels (except wildcard)
SETFPRIV	run *setfpriv*, set privilege vectors on files
SETFSLBL	run *setfslbl*, set filesystem minimum and maximum labels
SETPRLBL	run *setprlbl*, change current process security label
SETUID	run *setuid*, *setgid* to change real uid/gid of process
SETUNAME	run *setuname*, sets node name
SHMCTL	run *shmctl* to own all shared memory
SHUTDN	run *shutdn*, flushes I/O buffers and halts CPU
SIGNAL	run *signal* to catch a SAK signal and, in effect, disable the SAK.
STIME	run *stime*, sets the system time
ULIMIT	run *ulimit* to increase file size limit
UTIME	change modification and access times on non-owned files
VHANGUP	run *vhangup*, revokes all access to current control terminal
WILDCARD	allows process that also has SETFLBL privilege to set file labels to wildcard

Table 2.15. GPM privileges in Trusted XENIX

System Call	**Function**	**Privilege**
acct	enable/disable process accounting	ACCT
audit	enable or disable auditing	AUDIT
auditlog	append record to audit log	AUDITLOG
chroot	change the root directory	CHROOT
lock	lock a process in primary memory	PLOCK
mknod	make directory, or ordinary, special file	MKNOD
mount	mount a file system	MOUNT
plock	lock process, text, or data on memory	PLOCK
setflbl	set file label	SETFLBL
setfpriv	set file privileges	SETFPRIV
setfslbl	set file system labels	SETFSLBL
setprlbl	set process label	SETPRLBL
setuname	set node name of current system	SETUNAME
shutdn	flush block I/O and halt the CPU	SHUTDN
stime	set the time	STIME
umount	unmount a filesystem	MOUNT
uname	get node name of current system	SETUNAME
vhangup	virtually hangup the current control terminal	VHANGUP

Table 2.16. Privileged System Calls

System Call	**Function**	**Privileged Operation**
aclcreat	create an ACL	run on non-owned files
aclopen	open an ACL to operate on it	run on non-owned files
aclquery	fetch an entry from an ACL	run on non-owned files
aclrm	remove an ACL from the system	run on non-owned files
chmod	change mode of file	run on non-owned files
chown	change owner/group of file	run on non-owned files
fork	create new process	exceed limits
kill	send kill signal to process	run on processes without matching uid
link	link new file to existing file	link to a directory
msgctl	provide message control operations	use command IPC_RMID, IPC_SET
nice	change priority of a process	specify negative increment value
proctl	control active processes	send to all processes (except 0, 1)
semctl	control semaphore operations	use commands IPC_RMID, IPC_SET
setgid	set group id	set for non-owned files
setuid	set user id	set for non-owned files
shmctl	control shared memory operations	can use commands IPC_RMID, IPC_SET
signal	specify action on receipt of signal	allow catch of SAK signal
ulimit	get and set user limits	increase process' size limit
unlink	remove directory entry	unlink a directory
utime	set file access and modification times	set for non-owned files

Table 2.17. Non-privileged System Calls with Privileged Options

component of the group tag is the group set–a list of groups consisting of the group itself and all other groups dominated by it. This list is strictly determined by the numerical portion of the group tag in comparison with the numerical portion of all other group tags. Taken together, the two components of the group tag explicitly represent the position of a given group within the power hierarchy.

The values defined for the groups on Trusted XENIX are given in table 2.18.

Role	**Power Number**	**Group Set**
TSP	3	{bin, ssa, audit, so, aa, user}
SSA	2	{ssa, so, aa, user}
AUDITOR	2	{audit}
SO	1	{so}
AA	1	{aa}
USER	0	{user}

Table 2.18. Power Hierarchy

The TSP's group tag is greater than that of every other user. An SSA's group tag dominates that of every other user except the TSP, the Auditor, and other SSAs. For example, in the case of the trusted process *passwd*, the TSP can change the password of any user, while those in the SSA role can change their own password and any other user's except the TSP's, the Auditor's, and other SSAs'. Unprivileged user group tags are zero by default. The library subroutine *chkpriv* implements the power hierarchy for Trusted XENIX. It determines whether a subject (invoking process) can be permitted to change the security relevant attributes of an object. The subject is granted permission if the subject is the owner of the object, or the subject is not the owner of the object and the "power" of the subject is greater than that of the object.

The WILDCARD Security Label

Trusted XENIX assigns security labels to all subjects and objects. The security label assigned is a two part tag (hierarchical level, category set) that may take on the values specified by the SSA in the clearance table */etc/security/s_clearance* and the category table */etc/security/s_category*. The WILDCARD label is defined in these tables as the security label whose hierarchical level equals "WILDCARD" and whose category set equals "none." It has a unique, specially defined, relationship to all other defined security labels. Specifically, the WILDCARD security label "equals" (and by extension, "dominates") any other defined security label, regardless of that label's clearance level or category set (comparison of labels is performed by the library routine *SLcmp*).

The motivation for defining the WILDCARD label in Trusted XENIX is to create a well-defined mechanism for setting up multilevel objects within the system without having to explicitly handle each instance with appropriate code within the kernel. Typically, these objects are TCB-maintained storage areas where information of differing security levels is stored in common. The following is a list of these objects:

- */dev/null*
- */dev/tty*
- */dev/swap*
- */dev/root*

- */dev/hd0A*
- */dev/kmem*
- */dev/mem*

A label may be set to the WILDCARD value only if the process attempting to set the label possesses the GPM privileges of SETFLBL and WILDCARD. Only the trusted process *init* possesses both of these privileges. The trusted process *mount* possesses the WILDCARD privilege and inherits the SETFLBL when it invokes the *mkdir* program. *mount* uses this privilege to create the */.ACL* directories, which will contain access control list (ACL) files. Since the ACL files can have security labels which differ from one another, the containing directory (*/.ACL*) must have the WILDCARD label. In addition, the TSP, when running in maintenance mode, has all the privileges of the trusted process *init*. The TSP operating in the maintenance mode can therefore set the WILDCARD label on any file system object.

2.8.4 Auditing

Control of auditing in Trusted XENIX is accomplished through a combination of actions performed by the TSP and the Auditor. The TSP defines the names of auditable events during system installation and the Auditor is responsible for correctly managing auditing once the system is running. Thus, only privileged users may control the audit mechanism.

Each audit event is assigned an audit level (the event audit level), and each user is assigned an audit level (the process audit level). Process audit levels can be zero, one or two; event audit levels can be zero through three. An event is audited if the event audit level is greater than the process audit level. Thus, a value of zero means an event is not audited and a value of three means it will always be audited.

Process audit levels are set by the *login* trusted process for ordinary users, according to the user and group specified at log in. It is set to the lower of the user and group process audit levels. The default values are one for user audit levels, and two for group audit levels. Processes that are created other than through *login* (such as daemons that are created during initialization) have a fixed process audit level of one.

To illustrate the use of this audit level mechanism, consider a call to *mkdir*, which issues two system calls to *link*. If the Auditor set the audit level of *link* to two and *mkdir* had its audit level set to two, then *link* will not generate an audit record when called from *mkdir*. If the Auditor instead set the audit level of *link* to three, calls to *link* from *mkdir* would then be audited. In this manner the generation of redundant audit records can be controlled. In addition, *mkdir* generates its own audit events.

The TSP defines the names of auditable events during system installation through the use of the *auditnam* command. The TSP also sets the audit level for each event, turns on auditing and defines the directory which will contain the audit trail. This directory has owner *audit*, group *audit* and protection mode 750, meaning that a process with uid *audit* has "rwx" access, one with gid *audit* has "rx" access, and all others have null access to the directory. This directory is also defined at system high (the maximum possible sensitivity level, and all categories.) The name of the active audit trail resides in the file */etc/security/audit/log* and the names of all previous audit trails reside in the file */etc/security/audit/pastlog*.

The protection modes on the audit trail are "rw" for the TPs with uid *audit* and "r" for the group *audit*. The maximum size of the audit trail is settable by the Auditor.

Through the *auditsh* set of commands, the Auditor has the capability to start and stop auditing, set the active event levels to those in the audit control file, change the audit event levels, delete old audit trails, display the current audit trail, change the audit trail size, and change the table of object security level ranges. The Auditor is also provided with a command, *c_chaudlvl*, which allows changing of process audit levels. All Auditor commands are fully audited.

Through the use of an audit reduction tool and the standard *grep* command, Trusted XENIX has the capability for selectivity based on uid, gid, security level, and any other field within an audit record.

The actual audit logging mechanism is part of the TCB and is implemented by the kernel routine *sacct*. This routine collects information from the user block and kernel data areas and places it into the in-core auditing buffer. The auditing file is written in the same manner as any disk I/O performed inside the kernel. If the write is unsuccessful, messages are sent to the system console, auditing will stop and the system will shutdown. TPs use the *auditlog* routine, which is a privileged system call which uses the kernel routine *sacct* to generate an audit record. It can be invoked only by a process that has the AUDITLOG privilege.

The maximum amount of audit data that could be lost from a system failure is the amount of audit data generated in the time between writes of disk buffer data to the disk. This occurs at least every 30 seconds in the standard system configuration. A system administrator can specify a smaller time interval if they desire. For example, if the data were written once each second then only one second of audit data would be vulnerable to a system failure. The theoretical maximum amount of data that would be lost given an *sync* interval of 1 second was computed to be approximately 263 records; the actual maximum will be much less, and will be based on the amount of other system activity and the amount of audit data being generated at the time of system failure.

2.8.5 Object Reuse

Whenever a new storage object is allocated to a subject, the object either contains no information or it contains only information assigned to it by the kernel. Storage objects are allocated by the TCB as described below.

- FILE: The *creat* kernel call creates a file with zero length. Any attempt to read past the end of file fails. The *open* kernel call can create a file (if the `O_CREAT` flag is set); this is implemented internally by calling *creat* and behaves identically. A file system's superblock contains an array used to cache the numbers of free disk blocks in the file system. The utility program *mkfs* organizes the data blocks of the file system into a linked list. As data are written into a file (i.e., a file "grows"), the kernel allocates a buffer for a new data block and clears the buffer by overwriting it with zeros.

- DIRECTORY: The *mkdir* kernel call creates and initializes a directory. All directories initially contain two entries, "." and "..". Each entry comprises 16 bytes (2 for the inode number and 14 for the name of the entry), giving the newly created directory 32 bytes of information. Any attempt to read past the initial 32 bytes of the newly created directory fails. When a file is deleted from the directory, its inode number is set to zero and the filename is overwritten with blanks in the directory entry.

- PIPE: The *pipe* and *mknod* kernel calls create pipes. Normally a process waits until data are in the pipe before it reads from the pipe; if the pipe contains no data, no read occurs. If a process attempts to read a newly created pipe before any data are written, either the read will wait, or if the `O_NDELAY` flag is set for the read call, the read returns a zero, thus satisfying the Object Reuse requirement.

- DEVICES: Only a TP may directly access a device. For example, only the TP *star* is allowed to access the diskette drive (for the unprivileged user). Each TP is responsible for ensuring that devices are cleared before reuse. Users have direct access to the terminal, which is allocated to the user by *login*. Any attempt to read the terminal buffer upon allocation (i.e., before the user has written into it) returns zeros. All video buffers, and caching buffers on disk controller cards are overwritten with zeros by the drivers which control them before reuse.

- XENIX SEMAPHORES: The *creatsem* kernel call creates a zero-length file (if it does not already exist), writes a zero to it (since the semaphore will hold only a numeric value) and opens it (i.e., allows the creating process to use it). The *mknod* kernel call can be used to create a semaphore; however, it is not initialized until the *opensem* kernel call is issued. Since *mknod* creates a zero-length file, the Object Reuse criterion is met.

- XENIX SHARED-MEMORY SEGMENTS: The *sdget* kernel call with the SD_CREAT flag set creates a Xenix shared-memory segment (if it does not already exist) and attaches it to the data space of the current process. The kernel fills the shared-memory area with zeros, so a read will return zeros.

- SYSTEM V SEMAPHORES: The *semget* kernel call creates and gives access to a set of semaphores, initializes them to zero, and fills in the other fields with kernel information.

- SYSTEM V SHARED MEMORY SEGMENTS: The *shmget* kernel call creates a new region of shared memory or returns an existing one, and the *shmat* kernel call attaches a region to a process. The kernel overwrites with zeros memory allocated by the *shmget* call, so that a read to a newly allocated area returns all zeros.

- SYSTEM V MESSAGE QUEUES: The *msgget* kernel call creates a message queue (if it does not already exist) and returns a message descriptor that designates the message queue. Messages are on a linked list per descriptor. If a subject attempts to receive a message before any other process sends a message, no message is received from the queue.

- ACLS: The *aclcreat* kernel call creates a new ACL or initializes an existing ACL. In creating a new ACL, the call returns a descriptor to a zero-length file. If the ACL already exists, the kernel truncates it to length zero before returning the descriptor. In either case, an attempt to read past the end of the ACL file will result in an error.

- MAGNETIC MEDIA: The *star* TP provides a secure means of writing to and reading from the built-in diskette drive or cartridge tape unit (CTU). A general user can use *star* (only from the trusted shell) to back up files he or she owns. A user can read from a diskette or cartridge tape only files which he or she owns and can read them only into a directory which he or she owns. Thus, no information stored on the media by a previous subject is available to any subject that obtains access to the media by using normal *star* procedures.

- MEMORY: Memory is allocated when a new process is *fork*ed and *exec*ed. Memory is also allocated on swap in and out. When memory is allocated or extended for a new process, it is overwritten by the new process' data or explicitly with zeros (e.g., large non-initialized arrays). The same is true for swapping processes. Any attempt to read past the end of the process' allocated memory returns an error.

- CPU REGISTERS: All process visible 80286 registers are saved when (in the `u_area`) when a process swaps out and are restored (from the u area) when a process swaps in. Hence, no information is maintained across a context switch.

All process visible 80386 registers, except the extra FS and GS registers, are treated like the 80286 registers. The FS and GS registers are cleared during a context switch.

When a process that is, or has been, using the 80287 or 80387 math co-processor is swapped out, the main CPU issues an instruction to *wait* until the co-processor has completed any operation that might be in progress and then *reset* (i.e., clear and reinitialize the registers) that co-processor.

2.8.6 Trusted Path

A trusted communication path is provided between the user and the TCB. This path protects the user during initial log in and authentication or any time the user wishes to interact with the trusted shell.

To implement the trusted path, a mechanism called the Secure Attention Key (SAK) is used. The SAK mechanism consists of two `control-Z` characters pressed in quick succession (less than a second between them). This sequence of characters is detected within the device driver. When the trusted path is invoked, all of the invoking user's processes are immediately terminated. Both read and write access to the terminal are revoked, and the trusted shell is given control. The device driver sends a special signal, SIGSAK, to *init*. This signal causes the SAK signal handler in *init* to be executed. The signal handler reads the current entry for the terminal line and determines the previous state of the controlling process on the line. If it reflects a logged in user (even if in TSH_PROCESS state already), it changes the state to TSH_PROCESS and calls a routine to start the trusted shell (else a *getty* is spawned on the line). Since the TCB inspects all keystrokes before they are given to a process the SAK mechanism is prevented from being interrupted by an intruder process.

The trusted shell *tsh* will prompt the user for a command from a menu of trusted commands. These commands allow such things as changing passwords to listing files in the current directory. Users can be assured that they are communicating with the TCB and not a spoofing process.

When the trusted shell is invoked, the *init* trusted process writes a special value in file */etc/utmp* to indicate entry to the trusted shell via the SAK mechanism. Following this, *init* executes the trusted shell, which displays a menu of commands to the invoking user available under the trusted shell.

If a user attempts to invoke the trusted shell directly (no SAK mechanism), the special flag is not set, and the process state does not change to TSH_PROCESS. A message will be displayed informing the user that this is not a trusted communication path. No programs will perform their trusted functions from the trusted shell unless the trusted path is used.

2.8.7 Archiving

The Trusted XENIX Secure Tape Archiver (the *star* Trusted Process) writes files to and retrieves them from archival storage (i.e., diskettes or cartridge tapes). The *star* TP is capable of creating multi-level archives and can be invoked only while running under the trusted shell (or by the TSP in Maintenance mode). Only files owned by the user invoking *star* may be extracted or retrieved (the TSP and SO are exempted from this requirement), and *star* will retrieve files only into directories owned by the invoker and will not retrieve files which already exist in the file system. Files restored by *star* are restored to their original location in the file hierarchy (absolute pathnames are used).

The first action taken by *star* is to determine the caller's "state" by calling *chkTPstate*, which returns an integer value indicating the privilege state of the process which invoked *star*. If the privilege state is something other than trusted shell, restricted trusted shell (a state available only to the SO), or maintenance mode (a state available only to the TSP), *star* terminates without performing any action. When called by a normal system user, *star* checks file ownership against the process uid and allows file exportation only when the two are the same. When called by either the TSP or the SO, *star* operates in a privileged mode, running with MAC_EXEMPT and DAC_EXEMPT privileges.

File attributes (e.g., protection bits, ACL, security label, uid, gid, size, modification time) are preserved in the archive copy. However, the setuid, setgid, and "sticky"[25] bits are preserved when *star* is invoked by the SO or TSP, but not when it is invoked by an unprivileged user. GPM vector bits are always set to zeros by *star*. TSP action is necessary to restore GPM privileges to privileged programs. However, since all privileged programs are registered in *s_install*, the TSP can refer to this file to determine what privileges should be restored to files archived using *star*.

Files exported by *star* are encrypted using Data Encryption Standard chain-block encryption if the SSA has installed an eight-byte key in */etc/security/data/star_key*. This key is accessible only to the SSA. The normal user is not informed of the key value or allowed to select or turn off encryption of archived information. The normal user is, however, informed as to whether the SSA has encryption enabled or disabled. Integrity of information retrieved by *star* can be enhanced only if the encryption has been enabled. The TFM specifically recommends that the SSA always set the key to a non-null value.

Invoking the trusted shell and running *star* is the only way an unprivileged user can export information to the diskette or cartridge tape drives. Since it is exempted from MAC and DAC checks, *star* allows the user to dump and retrieve an entire file hierarchy to or from a medium, as long as the security levels of exported files are bracketed by the device minimum and maximum levels of the diskette drive, as specified by the SSA. The user is not informed of the range of security levels or the highest level of the files exported or imported by *star*.

The standard UNIX routine Tape Archiver (*tar*) is available, but it does not allow unprivileged users to access the diskette or cartridge tape drives, since neither *tar* nor unprivileged users possesses the access permissions to the drive special files (device drivers). The checksum for *star* is the one's complement of the checksum used in *tar*, thus helping to ensure that *tar* and *star* files do not become confused.

Both the TSP and the SO can use *star* (while within maintenance mode or the privileged trusted shell, respectively) to export any file in the system. The *star* TP ensures that a file's security labels, ACLs, and original uids are also exported and properly represent the security level, access permissions, and ownership of the file.

2.8.8 Printer Spooler

User print requests are made using the system command *lp <pathname[s]>*. The first action performed by *lp* is to perform MAC and DAC checks (for read access) on the user's print request. The necessary calls to perform these MAC and DAC checks are made by *lp*, which passes the user's real uid to the routines that actually do the checks. At the time it is called, *lp* itself is operating with the effective uid of the pseudo-user *lp*.

If the MAC and DAC checks grant the required access, *lp* creates an entry in the printer spool directory

[25] The "sticky" bit is not part of the DAC mechanism and therefore has not been discussed here.

(/usr/spool/lp) indicating the user's work request. At that time, *lp* is still operating with the effective uid of *lp*. After it has entered the work request, *lp* changes its effective uid to that of the requester (leaving open the newly created file entry in the printer spool directory) and copies the requested file into the print spool directory, where it is stored with the user's work request and the user's CPL. Files to be printed are always copied into the printer spool directory at the time of the print request. The label associated with each file in the print spool directory is the CPL of the process which invoked *lp*, not the actual label of the file itself. The role of *lp* in printing the file is then complete, so it *exit*s and the user continues with his or her original process.

The change in *lp*'s effective uid is mandated by the nature of its MAC and DAC privileges (as defined by its GPM vector). So that it can violate the MAC policy (i.e., place files of differing security levels in the same directory), *lp* is MAC_EXEMPT. It is not DAC_EXEMPT, however, and must therefore operate with the effective uid of the user (while maintaining the gid of *lp*) to copy the user's file into the printer spool directory.

The trusted daemon *lpsched* polls the spool directory looking for work requests. Whenever it finds such a request, it *fork*s a copy of itself which will manage the print request and will, in turn, *exec* a print routine (*nwprt*) to perform the actual printing. The child copy of *lpsched* first checks the label associated with the file to be printed against the level associated with the printer itself. The label associated with the file to be printed is the CPL of the process requesting the print and must be within the range of security levels assigned to the printer by the SSA (in the *s_device* table). *lpsched* then accesses the */etc/security/s_category* and */etc/security/s_clearance* files to obtain the SSA-designated human-readable labels that correspond to the machine-readable security labels included with each file in the spool directory. *lpsched* schedules the individual printers, having sole access to their individual device drivers, and finally *exec*s *nwprt*, which controls pagination of the output file, inserts beginning and ending banner pages, and labels the top and bottom of each intermediate page (with the label of the file, or CPL of the requester if the input is not a file).

In the event the label stored with the file to be printed is not dominated by the Maximum Device Level of the requested device, *lpsched* prints a standard message (on the requested printer) indicating that the requested file cannot be printed. The only information specific to the print request that is contained in this message is the number of the print request. The process responsible for actually sending data to the printer is *nwprt*. *nwprt* has been modified for Trusted XENIX so that it will print a circumflex in place of any hex character that is less than 20 or greater than 7E. In addition, nwprt will print the following characters: 00 (null), 08 (backspace), 09 (horizontal tab), 0A (new line), 0C (form feed), and 0D (carriage return). In this way, no software re-configuration of the the printers in the evaluated configuration is possible through escape sequences in user data sent to the printer.

Banner pages cannot be overridden, and spoofing is prevent by means of a system-generated random number which is printed on the header and trailer banner pages. Labels placed at the top and bottom of each intermediate page can be overridden using an option to the *lp* command. The default for intermediate page labeling is "on." Enough room on the banner pages is reserved for the longest possible label. All print requests are auditable. Disabling of intermediate page labeling is also auditable.

2.8.9 Deflection Mechanism

Trusted XENIX employs a deflection mechanism which supports hidden */tmp* directories. This directory contains files that are created for temporary purposes (e.g. while editing or compiling programs). Therefore the directory would have to be readable and writable by all subjects. However, since directories are single-level objects, this causes a problem.

Trusted XENIX solves this problem by assigning the system low security label to */tmp*, and creating a number of subdirectories within it, one per security level. The subdirectories are of the form "0x<map>", where map is a sequence of twelve hexadecimal characters that can be mapped to the appropriate security label of that subdirectory.

When a process is created, a flag called the deflection flag is set. The child process of a *fork* inherits the deflection flag of the parent process, and the transformed process of an *exec* keeps the deflection flag of the calling process. This flag tells *namei* to use deflection; i.e., convert the path from */tmp* to the appropriate subdirectory, when resolving pathnames that use */tmp*. The user has the option of turning off this flag with the *chtmp* command; if this is done, the deflection flag is not set, and *namei* will perform no deflection (i.e., references to */tmp* are conventional).

A hidden subdirectory of */tmp*, if non-existent, is created at log in time by the *login* trusted process. The mail utility also makes use of this deflection mechanism to provide a secure mail utility.

2.9 TCB Assurances

The following section will discuss the various assurance provided by Trusted XENIX, including maintenance and installation of the system, configuration management, the functional testing philosophy, and methods used to recover from system failure. This section will also include a general discussion on covert channel management.

2.9.1 Maintenance and Installation

The TSP is responsible for installing and maintaining the operating system in a manner that ensures that the security requirements of Trusted XENIX are satisfied. The system provides several tools and procedures for the TSP to accomplish this task.

Role of Trusted System Programmer

The TSP requires the highest skill level of the administrative users because this role involves generation, installation, maintenance, and system recovery. The TSP must be familiar with the basic design of the system including the kernel and trusted process architecture and the operation of the hardware components. In addition, the TSP role requires familiarity with the use of all tools for system installation, diagnostics, and recovery. The TSP role must be entrusted with the security and integrity of the system before the TCB is operational. In the normal operating mode, the TSP special functions cannot be performed (the system must be in maintenance mode).

Initial Installation

The TSP is responsible for installing the evaluated hardware properly using the guidelines described in the *IBM PC/AT Installation and Setup manual* [INST] or the IBM PS/2 *Quick Reference Guide*. After the hardware is installed, the TSP runs the "System Checkout" program to verify its integrity. Once the hardware is functioning properly, the TSP installs the software (operating system) following a step-by-step procedure (which includes initializing all the security-relevant data bases) detailed in the *Trusted XENIX System Administration Manual* [SYAD]. Finally, the trusted program *scheck* is run by the TSP to verify that the data base defining the security profiles of the system satisfies the security requirements. It also does a consistency check on the attributes of all the command and data files to ensure that they were set up correctly.

Using Diagnostic Software

When power is applied to the System Unit, the POST program is automatically executed. The POST resides in the IBM PC/AT, IBM PS/2, or PC/AT Clone ROM, and consists of 22 test programs. It contains diagnostic routines that test the system board, the memory (including readback checks of every byte of RAM), and the installed control units and I/O devices. It also tests standard operation codes, privileged instructions, interrupts and protection mechanisms. The POST eventually returns to the ROM boot program to continue the start-up procedure and attempts to load an operating system from the first diskette drive (drive `A`). If that drive does not provide a program to load, the system attempts to load from the first fixed disk (drive `C`). If this operation is not completed, the BASIC interpreter is made available to the user (see section 2.5.7, page 20).

Another diagnostic tool provided is called Advanced Diagnostic Tests which is on a diskette and, hence, can be run only by the TSP in maintenance mode. These stand-alone programs are menu driven and provide more detailed testing of system components.

There are no other on-line hardware diagnostic routines available for Trusted XENIX.

2.9.2 Configuration Management (CM)

All changes to Trusted XENIX are directed and controlled by the Configuration Control Board (CCB), which meets periodically or as needed. The CCB consists of four individuals: a chairperson and representatives from Marketing, Development, and Trust Engineering. The administrative functions involved in managing changes to the system are performed by the CM Administrator, and the CM Manager manages the CM process.

The CM process involves two types of reports:

- A Product Change Request (PCR) is used to document a proposed change to the existing product (hardware, software, and/or documentation).
- A Change Tracking Report (CTR) is a document used as the central means of tracking progress of a change from the PCR-assessment stage through change implementation and testing.

Anyone involved with the product may submit a PCR to the CM Administrator, who assigns the PCR a number, logs it, and arranges for presentation to the CCB. The originator presents it to the CCB, which either defers it, rejects, it, or assigns it to an assessor, in which case one of the first responsibilities of the assessor is to draft a CTR for the PCR.

Once the PCR has been assessed, it is reviewed by the CCB, along with its draft CTR. At this point, the CCB may either reject it (in which case, both the PCR and draft CTR are filed), defer it for consideration at a later meeting, call for a reassessment, or accept it. If it is accepted, the CM Administrator closes and files the PCR and logs the CTR, and an implementor is assigned the task of drafting the implementation of the change.

To implement the change, the implementor obtains copies of all necessary configuration items from the CM Library, which is used by the CM Administrator to manage all Configuration Items (CIs). All CIs affected by a change are identified in the CTR package. The implementor drafts the necessary changes and performs unit-level testing; upon successful completion of unit testing, the changed CIs are submitted for integration and security testing. If any additional problems requiring further investigation arise during implementation, the implementor drafts and submits a PCR regarding the changes needed.

When the change has passed the trial integration and security testing, the CTR package is presented to the CCB for final review, at which time it may be accepted, deferred, or rejected. If accepted, the implementation is formalized by checking the modified CIs into the CM Library and closing and filing the CTR. The decision to package the implemented change or group of changes into a product release is outside the responsibility of the CCB. Deferred CTRs remain open for future CCB review; rejected CTRs are closed and filed.

2.9.3 Functional Test Software

The system is tested to assure that the security mechanisms and related software work as expected. Most testing occurs at the TCB interface, with additional testing of security mechanisms. All user-visible trusted processes and kernel calls directly available at the TCB interface are tested against their Descriptive Top-Level Specification (DTLS). A "grey box" testing methodology is used to test the kernel calls. It is a method of achieving the thoroughness of black box testing without the excess verbosity and without sacrificing test coverage. For more details on the functional test software, see section 3.24 on page 99.

2.9.4 Recovery From System Failure

When Trusted XENIX encounters a situation that indicates a hardware failure or an inconsistency in critical TCB data structures an automated impact assessment is made. If the damage is such that it cannot be repaired without human intervention and the situation is such that Trusted XENIX cannot continue without possibly causing more damage or resulting in a policy violation, the TCB halts with an error message which is printed on the system console. In order to proceed one must try to reboot the system after correcting the error indicated by the error message. In many cases, this will require the intervention of the TSP.

When the system is repaired to the point that one believes it will boot again, an authorized user should power-cycle the system thus causing the system to begin the normal boot sequence. During the normal boot sequence, the system will execute the program *scheck* and the shell script */etc/rc*, which contains a list of commands that the system administrator selects to be automatically executed. TIS distributes the system with an example */etc/rc* which contains the *fsck* command.

The *scheck* command ensures that protection-critical data structures are in a consistent state. For example, *scheck* ensures that all the command and data files required for secure operation of the system exist and are consistent with each other (e.g., users are registered only on valid groups, users have clearances that are valid, printable label names exist for all defined levels), that all privileged programs in the Installation Table exist and that no other privileged programs exist, and that the file system satisfies the mandatory compatibility rule. If *scheck* detects an inconsistency, it will abort the boot. If no errors are detected, it simply returns, allowing the system to continue initialization with */etc/rc*.

When *fsck* is called with no arguments, it checks to see if the system was shutdown properly. If the system was shutdown properly, it simply returns. If it determines that the file system is possibly in an inconsistent state due to improper system shutdown (i.e., not all memory-resident file system data structures were written to disk before shutdown), it will begin a careful check of the file system. The *fsck* command is able to repair most file system damage it detects; however, in those rare cases when it cannot repair a file, it "moves" the file to the system directory *lost+found*. Files in this directory are inaccessible to all users except the TSP. When *fsck* has completed repairing the file system, it simply returns and the next command in */etc/rc* is executed. If the damage is such that *fsck* cannot repair it and cannot move the damaged file(s) to the directory *lost+found*, then the boot is aborted.

2.9.5 Covert Channel Analysis

TIS has identified covert channels of three different types: resource exhaustion channels, event count channels, and MAC conflict channels.

Resource exhaustion channels are covert storage channels that result from the ability of a process to detect the exhaustion of some finite shared resource. For example, in Trusted XENIX a single system table shared by all processes identifies all open files. Since this table is limited in size, a higher-level process could open enough files to fill the table. When the table is full, processes at all levels receive an error message to that effect and are prevented from opening requested files.

Event count channels are covert storage channels that result when a process is able to sense a change in a globally shared system resource which is caused by another process. For example, the total amount of available disk space in the system is available to processes at any level. This enables a process at a higher level to signal a lower level process by allocating and releasing disk space.

MAC conflict channels are covert storage channels that are caused by the conflict between the standard XENIX interface and the mandatory security policy being added with Trusted XENIX. For example, the standard XENIX interface for *rmdir* will allow deletion only of empty directories (those that contain no links to other directories or files). The mandatory security policy of Trusted XENIX requires the current process level to be equal to the level of the directory containing the directory that one desires to delete. This creates a conflict which results in a covert storage channel when one attempts to delete an upgraded directory, since the lower level process will be able to sense whether the upgraded directory is empty.

TIS performed a thorough analysis of the code[26] to identify covert storage channels. This analysis is described in the vendor's report [CCAID]. The analysis is based on systematically finding all of the resources in the system and identifying all methods of accessing those resources. This information is then analyzed to determine whether it describes any method capable of passing information in violation of the system's

[26] This analysis is based on analysis of an earlier version of the kernel, which was conducted by IBM. An incremental additional analysis was conducted by both vendors.

mandatory security policy. TIS's analysis includes following the calling tree of every kernel call and identifying the information flow within the kernel via parameter and variable tracing. By first defining the information flow properties of every C language statement, this analysis is designed to find every covert storage channel.

TIS identified 18 resource exhaustion channels, eight event count channels, and five MAC conflict channels. For each identified channel, an exploitation scenario was developed, and a maximum channel signalling rate calculated, assuming that primitive operations used in the exploitation of the channel are executed on the hardware model with the shortest execution times. In developing these scenarios, TIS made assumptions (about coding schemes and signalling strategies) that the team feels truly maximize the channel rates. The strategies used in estimating and then reducing the bandwidths of the covert channels is given in a vendor report [CCABW].

TIS has eliminated some of the event count channels found during the analysis by use of randomization in assigning values to variables that are globally visible (such as the pid), or by establishing separate number spaces for distinct security levels. TIS has inserted delays into some of the system calls which are used in the exploitation of some channels, and has set the delays to values such that the rates of all channels are less than ten bits per second.[27]

[27]For one pair of channels, for which the maximum rate calculation is about 25 bits per second, the team has accepted the vendor's argument that the calculation is based on unrealistic extremes, and that a sustainable rate using either of these channels is less than ten bits per second.

This page intentionally left blank

Chapter 3

Evaluation as a B2 System

3.1 Discretionary Access Control

Requirement

The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., access control lists) shall allow users to specify and control sharing of those objects, and shall provide controls to limit propagation of access rights. The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorized access. These access controls shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object. Furthermore, for each such named object, it shall be possible to specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given. Access permission to an object by users not already possessing access permission shall only be assigned by authorized users.

Applicable Features

The Trusted XENIX TCB defines and controls access between named users and named objects by using two enforcement mechanisms: ACLs and protection bits (which provide self/group/public control). The System Overview section on DAC (see section 2.8.1) provides a detailed description of these mechanisms and the algorithm applied to their use. The protection-bit mechanism allows the creator of an object to specify read, write, and execute accesses for him/herself, system-defined groups of users, and everyone else in the system. While specifying access to the granularity of a single user is possible using this mechanism, a far more convenient method is provided by the ACL mechanism. In addition to the read, write, and execute accesses which the protection-bit mechanism provides, ACLs allow a user to specifically grant "no access" to an individual user, group, or everyone.

The system-wide protection-bit default grants access only to the object creator and privileged processes. Users may override the system default by creating a protection-bit default for themselves. No method of specifying a default ACL is provided. Rather, a user can create an ACL by using the *acl mk* command, which duplicates in an ACL file the accesses specified in the object's protection bits. When an ACL is created for an object, the nine low-order protection bits for that object are set to zero.

Access permission to an object by users not already possessing access permission can be assigned only by the object's owner.

Conclusion

Trusted XENIX satisfies the B2 Discretionary Access Control requirement.

Additional Requirement (B3)

The following changes are made in this requirement at the B3 level:

CHANGE: The enforcement mechanism **(e.g., access control lists)** shall allow users to specify and control sharing of those **objects**. These access controls shall be capable of **specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object.**

ADD: Furthermore, for each such named object, it shall be possible to specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given.

Applicable Features

Trusted XENIX's ACL mechanism allows the creator of an object to grant access to named individuals and any number of system-defined groups of individuals, and to specifically deny access to named individuals and named groups.

Conclusion

Trusted XENIX satisfies the B3 Discretionary Access Control requirement.[1]

3.2 Object Reuse

Requirement

All authorizations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the TCB's pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system.

Applicable Features

Trusted XENIX ensures that all authorizations to information contained within its storage objects, as well as system memory, are revoked prior to initial assignment, allocation, or reallocation to a subject. The Object

[1]Although Trusted XENIX satisfies this requirement at the B3 level, it does not satisfy the assurance requirements above its rated level.

Reuse Implementation section of the System Overview (section 2.8.5) provides a detailed description of the mechanisms which provide this assurance.

For all file-type objects (user files, directories, pipes, etc.), a zero-length file is created (for directories and pipes, the kernel performs some initialization), and any attempt to read past the end of the file fails. Reading the file produces the initialized data, or an end-of-file indicator for a zero-length file. As data are written into the file, the kernel allocates individual disk blocks as needed and clear the buffer used to store the data to be written into the block. For shared memory segments, upon allocation and as the area is extended, the kernel overwrites the allocated area with zeros. Devices are accessed only by TPs or the kernel (except for terminals), both of which are trusted to clear device buffers prior to reuse. The terminal is the only device that is directly accessible to untrusted subjects, and the terminal driver (i.e., *init*) clears all data from terminal buffers between successive users.

Process memory is overwritten at process creation, thus destroying existing data and ensuring that no data stored in memory by the prior subject's actions are available to the new subject. Any attempt to read past a process boundary (or descriptor boundary) will result in an error. In the cases where memory is not automatically cleared, the kernel ensures that the data in the memory are overwritten with zeros on process reinitiation.

Video devices and the console keyboard are restored to a known, fixed state when the device is released (i.e., the console user logs out), and all related buffers are pusrged.

Conclusion

Trusted XENIX satisfies the B2 Object Reuse requirement.

3.3 Labels

Requirement

Sensitivity labels associated with each ADP system resource (e.g., subject, storage object, ROM) that is directly or indirectly accessible by subjects external to the TCB shall be maintained by the TCB. These labels shall be used as the basis for mandatory access control decisions. In order to import non-labeled data, the TCB shall request and receive from an authorized user the security level of the data, and all such actions shall be auditable by the TCB.

Applicable Features

The use of labels in Trusted XENIX is fully described in the System Overview section of this report (see section 2.8.2, page 58). All subjects and objects in Trusted XENIX are assigned sensitivity labels, which reflect the security level of the subject or object. These are used as the basis for all MAC decisions. *star* will not import non-labeled information, so no one except the TSP using either *tar* or the DOS commands can import non-labeled data. The TSP is responsible for labeling the data when imported in this manner.

Conclusion

Trusted XENIX satisfies the B2 Labels requirement.

3.4 Label Integrity

Requirement

Sensitivity labels shall accurately represent security levels of the specific subjects or objects with which they are associated. When exported by the TCB, sensitivity labels shall accurately and unambiguously represent the internal labels and shall be associated with the information being exported.

Applicable Features

For subjects, the *login* TP sets the CPL and the terminal level at log in time (see section 2.7.1, page 44). The user can change the CPL only by logging out and reaccomplishing log in.

Security labels are associated with inodes (file system objects), ipc objects, and processes by making an entry in the corresponding structure for each. For example, the CPL of a process is stored in the `proc` structure within the `proc` table. These labels are accessible only through kernel or trusted processes and are therefore protected through the use of the protection mechanisms discussed in the Trusted Process section of the overview (see section 2.6.6, page 33).

All created objects, except upgraded directories, inherit the level of the creating process. New files may be installed only in a directory that has the same level as the creating process. New directories can be created at a level equal to or greater than the level of the parent directory using the *mkdir -s* command. The child process resulting from a *fork* inherits the parent's security level. The *exec* system call does not change the CPL.

Labels exported by the *star* TP are stored with the archived file.

Conclusion

Trusted XENIX satisfies the B2 Label Integrity requirement.

3.5 Exportation of Labeled Information

Requirement

The TCB shall designate each communication channel and I/O device as either single-level or multilevel. Any change in this designation shall be done manually and shall be auditable by the TCB. The TCB shall maintain and be able to audit any change in the current security level or levels associated with a communication channel or I/O device.

Applicable Features

Hard disks, floppy diskettes, cartridge tapes, and printers are implemented as multilevel devices in Trusted XENIX. The range of security levels allowed for a particular drive or printer is given in a specific entry for that device in the system security file */etc/security/s_device*. This range is settable only by the SSA and only through the SSA's restricted trusted shell. Any change in these SSA defined ranges is auditable.

Terminals in Trusted XENIX are implemented as single-level devices. The security level of the terminal is realized as the security label of its inode (actually, that of the special file representing the terminal) and cannot be set, handled, or otherwise interpreted by the terminal driver.

A terminal operates at a single level throughout a user session. This level can be changed only by the user physically logging out and back in, creating a new session, and causing an audit record to be generated. This level is determined by the trusted process *login* at log in time. A user may query the system at any time during the session to cause the system to display the current session level. SSA definition of a terminal minimum and maximum levels is the same as that described for printers, and the levels specified are stored in the same security table.

Conclusion

Trusted XENIX satisfies the B2 Exportation of Labeled Information requirement.

3.6 Exportation to Multilevel Devices

Requirement

When the TCB exports an object to a multilevel I/O device, the sensitivity label associated with that object shall also be exported and shall reside on the same physical medium as the exported information and shall be in the same form (i.e., machine-readable or human-readable form). When the TCB exports or imports an object over a multilevel communication channel, the protocol used on that channel shall provide for the unambiguous pairing between the sensitivity labels and the associated information that is sent or received.

Applicable Features

Trusted XENIX provides two forms of output to multilevel devices: printer output and files archived to or recovered from diskette or cartridge tapes. Both of these are described in detail in the System Overview (see section 2.8.7, page 68 and section 2.8.8, page 69). Exportation requirements for the printer subsystem are discussed in section 3.8, page 83.

Except for the SSA, all users must use the *star* TP to access the diskette drives (*/dev/fd0* and */dev/fd1*) and the cartridge tape drive (*/dev/wtr*). Since the device special files are owned by group *remmed*, they are only accessible to the SSA through *star*. The TSP may also access the device special files through *tar* or direct copy routines.

Conclusion

Trusted XENIX satisfies the B2 Exportation to Multilevel Devices requirement.

3.7 Exportation to Single-Level Devices

Requirement

Single-level I/O devices and single-level communication channels are not required to maintain the sensitivity labels of the information they process. However, the TCB shall include a mechanism by which the TCB and an authorized user can reliably communicate to designate the single security level of information imported or exported via single-level communication channels or I/O devices.

Applicable Features

Terminals in Trusted XENIX are implemented as single level devices. The terminal may be logically viewed as consisting of the physical terminal, the physical port adapter, the terminal special file, and the associated line discipline module. The security level of the terminal is realized as the security label of the terminal special file's inode. This label is initially set to the appropriate level at log in time by the trusted process *login*. The terminal is not trusted to set, handle, or otherwise interpret labels on its own. Access mediation to the terminal is performed by the access mediation module within the kernel.

Minimum and maximum device labels for terminals are settable only by the SSA and are used to constrain the terminal (device) level at log in. The terminal level can be changed only by logging out and back in. This creates a new session and generates an audit record.

Conclusion

Trusted XENIX satisfies the B2 Exportation to Single-Level Devices requirement.

3.8 Labeling Human-Readable Output

Requirement

The ADP system administrator shall be able to specify the printable label names associated with exported sensitivity labels. The TCB shall mark the beginning and end of all human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly [2] represent the sensitivity of the output. The TCB shall, by default, mark the top and bottom of each page of human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly represent the overall sensitivity of the output or that properly represent the sensitivity of the information on the page. The TCB shall, by default and in an appropriate manner, mark other forms of human-readable output (e.g., maps, graphics) with human-readable sensitivity labels that properly represent the sensitivity of the output. Any override of these marking defaults shall be auditable by the TCB.

Applicable Features

The operation of the Trusted XENIX printer spooler subsystem is described earlier in this report (see section 2.8.8, page 69). The SSA can specify the human-readable (printable) labels that correspond to the machine-readable sensitivity labels associated with exported files. This mapping is specified by the SSA in the system files */etc/security/s_clearance* and */etc/security/s_category*. The printer spooler subsystem schedules the individual printers within the Trusted XENIX system (having sole access to their individual device drivers), controls pagination of output files, inserts beginning and ending banner pages, and labels the top and bottom of each intermediate page with the CPL of the job requester. (The CPL dominates the actual sensitivity of the printed file.) Labeling of banner pages cannot be overridden. Labels placed at the top and bottom of each intermediate page can be overridden using an option of the *lp* command. Intermediate page labeling is enabled by default. All print requests are auditable. Disabling of intermediate page labeling is also auditable.

Conclusion

Trusted XENIX satisfies the B2 Labeling Human-Readable Output requirement.

3.9 Mandatory Access Control

Requirement

The TCB shall enforce a mandatory access control policy over all resources (i.e., subjects, storage objects, and I/O devices) that are directly or indirectly accessible by subjects external to the TCB. These subjects and objects shall be assigned sensitivity labels that are a combination of hierarchical classification levels and non-hierarchical categories, and the labels shall be used as the basis for mandatory access control decisions. The TCB shall be able to support two or more such security levels. The following requirements shall hold for all accesses between all subjects external to the TCB and all objects directly or indirectly accessible by these subjects: A subject can read an object only if the hierarchical classification in the subject's security level is greater than or equal to the hierarchical classification in the object's security level and the non-hierarchical categories in the subject's security level include all the non-hierarchical categories in the object's security

[2] The hierarchical classification component in human-readable sensitivity labels shall be equal to the greatest hierarchical classification of any of the information in the output that the labels refer to; the non-hierarchical category component shall include all of the non-hierarchical categories of the information in the output the labels refer to, but no other non-hierarchical categories.

level. A subject can write an object only if the hierarchical classification in the subject's security level is less than or equal to the hierarchical classification in the object's security level and all the non-hierarchical categories in the subject's security level are included in the non-hierarchical categories in the object's security level. Identification and authentication data shall be used by the TCB to authenticate the user's identity and to ensure that the security level and authorization of subjects external to the TCB that may be created to act on behalf of the individual user are dominated by the clearance and authorization of that user.

Applicable Features

The TCB enforces a MAC policy (as described in TIS's *Interpretation of the Bell and LaPadula Model for Trusted Xenix* [INTP]) over all subjects, objects, and I/O devices that are directly or indirectly accessible by subjects external to the TCB. The System Overview section on Mandatory Access Control gives a detailed description of the MAC policy enforced by Trusted XENIX (see section 2.8.2).

To enforce this policy, all subjects and objects are assigned sensitivity labels that are a combination of one of up to 255 (plus the WILDCARD) hierarchical clearance or classification levels and a set of up to 64 non-hierarchical categories. Trusted XENIX therefore exceeds both the required (2 or more) and suggested (16 or more) numbers of hierarchical levels and meets the suggested number of non-hierarchical categories (64 or more), as specified in "A Guideline on Configuring Mandatory Access Control Features" in the TCSEC. The sensitivity labels are used as the basis for MAC decisions. The rules for MAC, as described in TIS's interpretation [INTP], meet the access requirements specified in the B2 Mandatory Access Control criterion above for accesses between all subjects external to the TCB and all objects directly or indirectly accessible by these subjects. Identification and authentication data are used by the TCB to authenticate the user's identity and to ensure that the security level and authorization of subjects created (external to the TCB) to act on behalf of the user are dominated by the clearance and authorization of that user.

Conclusion

Trusted XENIX satisfies the B2 Mandatory Access Control requirement.

3.10 Subject Sensitivity Levels

Requirement

The TCB shall immediately notify a terminal user of each change in the security level associated with that user during an interactive session. A terminal user shall be able to query the TCB as desired for a display of the subject's complete sensitivity label.

Applicable Features

Immediately following identification and authentication, the *login* TP sets the CPL for the user session. The CPL is composed of the hierarchical clearance level and non-hierarchical category set and is displayed on the terminal screen at the time of a successful log in. It will remain the user's current security level and may not change within a single Trusted XENIX session. Change of security level can be accomplished only by logging out and reaccomplishing log in. At any time during an active Trusted XENIX session, the user may query the system for the CPL using the *c_attr* command. The system will respond with a display of the user's complete CPL.

Conclusion

Trusted XENIX satisfies the B2 Subject Sensitivity Levels requirement.

3.11 Device Labels

Requirement

The TCB shall support the assignment of minimum and maximum security levels to all attached physical devices. These security levels shall be used by the TCB to enforce constraints imposed by the physical environments in which the devices are located.

Applicable Features

Trusted XENIX provides user interfaces to four types of devices—printers, terminals, diskette drives and cartridge drives. The SSA defines the allowable range for device levels at system generation time through specification of data structures located within the system file */etc/security/s_device*. The SSA assigns a maximum and minimum level for each file system and each terminal in the system (TMaxL, TMinL). The SSA specifies the security level range of devices by setting PDMinL and PDMaxL or SDMinL and SDMaxL. There is no effective distinction between private and shared devices.[3] PDMaxL, PDMinL, SDMaxL and SDMinL must be specified for each individual printer in the system. Printer output is constrained by the TCB so that the CPL of the job requester must be within the range of security levels assigned to the printer by the SSA.

Terminal Maximum Level (TMaxL) is used by the *login* process to constrain the CPL. The CPL must be dominated by TMaxL. The Terminal Minimum Level (TMinL) is used only to constrain the user and not the level of the process (CPL) run at the terminal in question. A user whose maximum level (UML) exceeds TMinL and whose minimum level is less than TMinL, is permitted to log in at a level below TMinL (all other checks being met).

Export to diskette and cartridge drives is accomplished through *star*. The *star* TP ensures that the security levels of exported files fall within the range set for the diskette or cartridge drive.

Conclusion

Trusted XENIX satisfies the B2 Device Labels requirement.

[3] There was a difference between shared and private in earlier versions.

3.12 Identification and Authentication

Requirement

The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall maintain authentication data that includes information for verifying the identity of individual users (e.g., passwords) as well as information for determining the clearance and authorizations of individual users. This data shall be used by the TCB to authenticate the user's identity and to ensure that the security level and authorizations of subjects external to the TCB that may be created to act on behalf of the individual user are dominated by the clearance and authorization of that user. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. The TCB shall be able to enforce individual accountability by providing the capability to uniquely identify each individual ADP system user. The TCB shall also provide the capability of associating this identity with all auditable actions taken by that individual.

Applicable Features

The *login* trusted process of Trusted XENIX is responsible for the identification and authentication of users before they are allowed to use the system. This interface allows users to identify themselves to the system by supplying a user name and password. Users may provide a group name and a security label (or be given default values) that allows them certain privileges. This procedure is described in the Security Features User's Guide [SFUG].

The password field in the file */etc/passwd* (the standard UNIX password file) has been removed in Trusted XENIX. In addition, group passwords and listed users have been removed from the group file. The passwords are stored in encrypted form in the user security profile, */etc/security/s_user*. This profile is indirectly accessible only by the SSA. No user can directly access this profile.

Passwords are user chosen. Each password must have at least six characters. If there are more than eight characters, only the first eight are significant, and they must contain at least two alphabetic characters (upper or lower case letters) and at least one numeric or special character. Each password must differ from the user's log in name and any reverse or circular shift of that log in name. New passwords must differ from the old by at least three characters. Finally, no character in the new password can be repeated more than once in sequence. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent. However, when an authentication check is made, it is case sensitive.

An authentication and authorization profile for each user is maintained by the TCB to grant access to various objects and to determine the security level of the subjects created by the user. This is accomplished by maintaining a user maximum security level and a group maximum security level. The UML comprises the user maximum clearance and the complete set of categories the user is authorized to access. Correspondingly, the GML defines maximum clearance and largest set of categories for a group of users. The user must specify a group name at log in or be assigned a default group with its attendant GML. The protection of this authorization data is achieved by requiring that all such data be part of the TCB and available only to the system security administrator and the administrator's trusted processes.

Each user is identified by a user ID and the associated uid, and a group ID and the associated gid. The uid and gid are both non-reusable identifiers. When a user or group is removed from the system, the uid and gid are not reassigned, thus preventing unauthorized access to any residual objects left by the removed user/group. At log in an audit record that associates the uid and gid of the user with a character string representing the userid and groupid is generated. Whenever a process is created, an audit record that associates a process ID (pid) with the uid and gid of the creator is generated. When that process performs an auditable event, the pid is included in the generated audit record. This provides a unique identity for all auditable events.

Conclusion

Trusted XENIX satisfies the B2 Identification and Authentication requirement.

3.13 Audit

Requirement

The TCB shall be able to create, maintain, and protect from modification or unauthorized access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorized for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers, and other security relevant events. The TCB shall also be able to audit any override of human-readable output markings. For each recorded event, the audit record shall identify: date and time of the event, user, type of event, and success or failure of the event. For identification/authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object and the object's security level. The ADP system administrator shall be able to selectively audit the actions of any one or more users based on individual identity and/or object security level. The TCB shall be able to audit the identified events that may be used in the exploitation of covert storage channels.

Applicable Features

After the initial set-up has been performed by the TSP, auditing in Trusted XENIX is invoked through the actions of the Auditor. An Auditor is a member of the privileged group, *audit*, who has the ability to perform special auditing functions through the trusted shell, *auditsh*. Auditing can be invoked in one of two ways: either through the Auditor's logging in using the trusted shell, or at system startup by the TSP's placing the *auditsh* command in the */etc/rc* startup file.

Auditing is implemented by assigning audit levels to users and kernel calls (see section 2.8.4 on page 65). Only the Auditor has the capability to set and change the audit levels.

The following three events must be audited in order for the audit post-processor to function properly:

1. Entry of the *exit* system call.

Auditing of this event allows the post-processor to know when a process has ended.

2. Exit of the *fork* system call.
   Auditing of this event allows the post-processor to know the parent process of a new process.
3. Internal and exit events of the *chdir* system call.
   Auditing of these events allows the post-processor to track the current working directory state of each process. Thus the post-processor can print full pathnames instead of relative path-names.

Additionally, the Auditor has the capability to specify auditing of the following events:

- System initialization actions
- All log in activity
- Change of password
- Accumulations of denied entry attempts
- Introduction of objects into address space
- Creation/deletion of subjects and objects
- SSA, AA, SO, and Auditor actions
- Denied access to objects and interprocess signals
- Access privilege distribution and revocation
- Overriding of page labels

Every audit record contains a fixed part and an event-dependent part. The fixed part of the record contains:

- Record sequence number
- Date and time
- Type of event
- Error number
- Return values
- Subject information (real and effective uid and gid, and the process ID)
- Security level of process or object

The event-dependent part may contain:

- Pathname of the object
- Parameters of the call

- Inode and file system numbers
- Text of Trusted Process message

Trusted XENIX provides an audit reduction capability which allows the Auditor to selectively retrieve data based on the following:

- uid and gid
- object identity
- event type
- pid
- object/subject security level
- user security level

The print function of the *audit* command allows for selection of audit records by uid and gid. Selectivity based on other items can be achieved through the combined use of the *audit* command and the *grep* command.

When the audit files are swapped, a message is sent to the console. A warning is generated and sent to the console if the amount of space requested by the auditor for a new audit file can not be allocated (the remaining space is, however, allocated). Once the file system becomes full and *audit* can no longer allocate space, then the system will shut down.

All covert channels identified by the vendor are exploited through kernel calls. As all kernel calls are auditable, events which may be used in the exploitation of covert channels are auditable.

Conclusion

Trusted XENIX satisfies the B2 Audit requirement.

3.14 Trusted Path

Requirement

The TCB shall support a trusted communication path between itself and users for initial login and authentication. Communications via this path shall be initiated exclusively by a user.

Applicable Features

The Secure Attention Key (SAK) mechanism provides a trusted path to the TCB. The SAK mechanism is implemented as two consecutive `control-Z` characters separated by less than one second. The TCB inspects all keystrokes before they are given to a process, thereby preventing the SAK mechanism from being intercepted by an intruder process. Once a trusted path is established, the user may proceed with the log in described in section 3.12 on page 86.

Use of the trusted path also revokes all access to the terminal from all processes, thus preventing a process from spoofing the trusted *login* process. Also, use of the trusted path while a user is logged in invokes the trusted shell and causes the TCB to identify itself to the user. Thus, if the logout process were spoofed, a subsequent invocation of the trusted path would result in this identification, alerting the user. While in the trusted shell, the user can safely invoke processes to do such things as change passwords, change file attributes and backup files.

Conclusion

Trusted XENIX satisfies the B2 Trusted Path requirement.

Additional Requirement (B3)

CHANGE: The TCB shall support a trusted communication path between itself and **users** for **use when a positive TCB-to-user connection is required (e.g., log in, change subject security level)**. Communications via this **trusted** path shall be **activated** exclusively by a user **or the TCB and shall be logically isolated and unmistakably distinguishable from other paths**.

Applicable Features

When the trusted path is invoked, a message is issued distinguishing this path from others, so that the trusted path is identified to the user. Trusted XENIX requires that security relevant actions be made through the trusted path. The commands implementing these actions will fail if the invoking user is not acting through the trusted path. In addition to each user having a trusted path, each special role (e.g., SSA) has a restricted trusted shell. This provides extra protection for the trusted processes used by these roles. The user activates the trusted path with two consecutive `control-Z` characters typed within one second. This results in an entry in a special file and a process attribute indicating a special state, checked by TPs before performing restricted actions. Users are instructed in the Security Features User's Guide (SFUG) to log in using the trusted path, and can not thereafter dynamically change sensitivity levels.

Conclusion

Trusted XENIX satisfies the additional provisions of the B3 Trusted Path requirement.[4]

[4]Although Trusted XENIX satisfies this requirement at the B3 level, it does not satisfy the assurance requirements above its rated level.

3.15 System Architecture

Requirement

The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). The TCB shall maintain process isolation through the provision of distinct address spaces under its control. The TCB shall be internally structured into well-defined largely independent modules. It shall make effective use of available hardware to separate those elements that are protection-critical from those that are not. The TCB modules shall be designed such that the principle of least privilege is enforced. Features in hardware, such as segmentation, shall be used to support logically distinct storage objects with separate attributes (namely: readable, writeable). The user interface to the TCB shall be completely defined and all elements of the TCB identified.

Applicable Features

The Trusted XENIX TCB (see section 2.4, page 7) is protected from external interference in differing ways depending on the part of the TCB in question. The "domain" of the TCB is actually a set of isolated domains: the kernel and the set of individual address spaces of the trusted processes. Kernel software is protected by the hardware privilege level mechanism: its code and data are accessible only when running at privilege level zero, and kernel software can be invoked only at a single well-defined entry point. The code run by trusted processes is protected by DAC on program files and by process isolation. The dynamic structures of trusted processes are protected by process isolation. Like the kernel, each trusted process may be invoked only by calling its well-defined entry point. Various restrictions (see section 2.6.6, page 33) are implemented to guarantee that an untrusted process cannot interfere with the operation of a trusted process. The data (files and directories) used by trusted processes are all protected by DAC, and some are protected by explicit use of MAC as well. Uncontrolled access to devices and memory is prohibited by DAC on the appropriate device drivers.

Process isolation is provided by giving each process an independent address space, both for directly accessible memory and for objects manipulated by system calls. Some process context is inherited at process creation (*fork*) but the information retained by each process cannot be further affected by actions of the other. Communication is possible only if both processes subsequently elect to share objects or use existing shared objects (such as pipes in existence at process creation). The *kill* system call can be used to signal only between processes with matching real or effective uids. A simple form of communication from child to parent process is provided by the child's completion code, but since the child of an untrusted (unprivileged) process has the same label as its parent, this is permitted by the MAC policy.

The Intel 80286 or 80386 privilege level and call gate mechanisms are used effectively to separate kernel code and data from all non-kernel process data. These hardware features are also used to achieve process isolation, thereby isolating each trusted process component of the TCB from the kernel and all other processes. A single Intel 80286 or 80386 call gate segment is used to mediate all system call invocations of the kernel. The code for each individual system call is responsible for making all access and privilege checks. Hardware access verification instructions are used by system calls to ensure the accessibility of their user-supplied arguments. A well-enforced programming discipline (see section 2.6.5, page 31) ensures that this mechanism is used consistently and protects against errors due to multiple references to user parameters. Trusted XENIX uses two of the four Intel 80286 or 80386 privilege levels (privilege levels one and two are unused).

The least privilege principle is enforced both within the Trusted XENIX TCB and in the administrative interfaces (see section 3.20, page 95). Within the TCB trusted processes are granted only the minimal privileges required to perform their functions and are designed so that these privileges are available only when necessary, by use of "privilege bracketing."

The kernel is divided in packages of source code containing C functions and data declarations. Each package is a file that has a "`.c`" suffix, and which contains various C compiler directives to allow collections of global data, type, and structure declarations to be included (i.e., as if they were declared in the source file itself) in the `.c` file. Each of these packages is a module. The question of whether a software system based on UNIX can meet the TCSEC class B2 assurance requirements has prompted considerable discussion. In the opinion of one study [SIBE], "bringing an existing system to the B2 level is likely to be at least as difficult as building a brand new system." Considerable effort was expended by TIS and IBM in restructuring the system, modifying interfaces, reducing the use of the `.h` files (`#include` files), and establishing general disciplines for manipulating common structures within the kernel. For the TPs, the vendors adopted a generic program structure (which includes verifying the execution environment of each TP on entry), established a mechanism for reducing the use of privileges, and enforced general programming practices for improving code readability.

An 80x86's segmented address space is used to isolate memory belonging to one process from that belonging to another, and to provide read and read/write protection on explicitly shared memory segments. However, because the UNIX kernel and process model is designed around the concept of a linear address space and a two-state protection architecture, code and data structures are not designed to be kept in independent segments. There is no significant use of segmentation within the kernel or within individual trusted processes, save for that minimally required to isolate processes, separate code from data, and accommodate the processors' segment size of 65,536 bytes. As described in section 2.6.4 on page 28, the limited segment size requires that large programs be organized to occupy multiple segments with identical security attributes, solely because 65,536 bytes is insufficient for many programs.

The Descriptive Top Level Specification (see section 3.18, page 94) provides a description of the user interface to the TCB, which includes the system calls, the user-invocable trusted processes, the trusted shell commands available through the trusted path, and the Intel 80286 or 80386 instruction set. The design documentation (see section 3.25, page 100) for the kernel and the trusted processes identifies all the TCB components and describes their interfaces.

Conclusion

Trusted XENIX satisfies the B2 System Architecture requirement.

3.16 System Integrity

Requirement

Hardware and/or software features shall be provided that can be used to periodically validate the correct operation of the on-site hardware and firmware elements of the TCB.

Applicable Features

As described in "Hardware Diagnostics", page 21, there are a number of diagnostic tests available for Trusted XENIX.

Each machine provides a set of power-on self-tests which test some minimal functionality of the hardware. Additionally, most of the machines comes with a set of Advanced Diagnostic Tests which provide more detailed testing, specifically of the peripherals.

Trusted XENIX also comes with a set of TIS created tests that are designed to test all of the security critical mechanisms of the base CPUs, including the ring mechanism, segmentation, control transfers, privileged instructions, and interrupt handling.

Finally, TIS has identified a utility, Check$\surd$It, that provides an extensive set of functional peripheral tests. This set is necessary to fill in both known and unknown deficiencies in the POST and advanced diagnostic test provided with the non-IBM hardware platforms.

Conclusion

Trusted XENIX satisfies the B2 System Integrity requirement.

3.17 Security Testing

Requirement

The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code, and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorization to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. The TCB shall be found relatively resistant to penetration. All discovered flaws shall be corrected and the TCB retested to demonstrate that they have been eliminated and that new flaws have not been introduced. Testing shall demonstrate that the TCB implementation is consistent with the descriptive top-level specification.

Applicable Features

The vendor and evaluation team conducted functional testing at TIS and found the system to work as claimed in the system documentation.

The penetration effort was conducted by the three-member evaluation team over a one week period of time at TIS. Activities during testing were all relative to the activities performed during the penetration testing effort for Trusted Xenix version 2.0 and included studying both documentation and code, generating hypotheses, testing flaw hypotheses, and creating tools to aid in testing (e.g., a program to read kernel memory). The hardware tested consisted of one of each major variant of the evaluated configuration.

The overall penetration effort was based on the flaw hypothesis method, whereby the penetration team meets and proposes possible system flaws. The hypotheses were assigned to individuals and were recorded in a log which was updated as progress was made and conclusions drawn. Each proposal was investigated, by thought experiment, documentation study, code study, or by coding a test scenario to exploit the flaw.

No flaws were identified for Trusted Xenix version 3.0. It should be noted that some non-security relevant flaws that existed in Trusted Xenix version 2.0 have been fixed in version 3.0.

Conclusion

Trusted XENIX satisfies the B2 Security Testing requirement.

3.18 Design Specification and Verification

Requirement

A formal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system that is proven consistent with its axioms. A descriptive top-level specification (DTLS) of the TCB shall be maintained that completely and accurately describes the TCB in terms of exceptions, error messages, and effects. It shall be shown to be an accurate description of the TCB interface.

Applicable Features

The Bell and LaPadula security model was interpreted for Trusted XENIX. The interpretation of the model states and state transitions in Trusted XENIX are defined, and the access control mechanisms of Trusted XENIX are shown to satisfy the Bell and La Padula axioms. The discretionary security and the activation axioms of Trusted XENIX are, in general, more restrictive[5] than those defined in the Bell and La Padula model. A comprehensive explanation of the model interpretation for Trusted XENIX is provided in *Interpretation of the Bell and La Padula Model in Trusted XENIX* [INTP].

The Trusted XENIX DTLS is provided in several manuals: *TIS Trusted XENIX Commands Reference Vols. 1 and 2* [CMREF 1], [CMREF 2]; *TIS Trusted XENIX System Reference* [SYSREF]; *iAPX 286 and 386 Programmer's Reference Manuals* [286 85], [386 86]. The system calls for the kernel and trusted processes that are visible at the user interface are described. The syntax is provided and the errors and exceptions are identified.

[5] The Trusted XENIX model is actually less restrictive in that revocation of access rights is not instantaneous.

Conclusion

Trusted XENIX satisfies the B2 Design Specification and Verification requirement.

3.19 Covert Channel Analysis

Requirement

The system developer shall conduct a thorough search for covert storage channels and make a determination (either by actual measurement or by engineering estimation) of the maximum bandwidth of each identified channel.

Applicable Features

TIS performed a thorough analysis of the Trusted XENIX to identify covert storage channels. This analysis is described earlier in this report.

Conclusion

Trusted XENIX satisfies the B2 Covert Channel Analysis requirement.

3.20 Trusted Facility Management

Requirement

The TCB shall support separate operator and administrator functions.

Applicable Features

Trusted XENIX supports separate operator and administrator functions by restricting the commands available to each role/position. This separation is implemented by establishing a unique, reserved group name for each role. These roles, described in section 2.3.2 on page 6, are: Trusted System Programmer, System Security Administrator, Auditor, Secure Operator, and Accounts Administrator. At log in time, the members of each role must choose the specific group name they are to be identified with for that session. For the remainder of the session, they are restricted to commands pertinent to that role. For example, no Secure Operator commands can change the security attributes of any protected objects.

Conclusion

Trusted XENIX satisfies the B2 Trusted Facility Management requirement.

Additional Requirement (B3)

The following changes are made to this requirement at the B3 level:

ADD: The functions performed in the role of a security administrator shall be identified. The ADP system administrative personnel shall only be able to perform security administrator functions after taking a distinct auditable action to assume the security administrator role on the ADP system. Non-security functions that can be performed in the security administration role shall be limited strictly to those essential to performing the security role effectively.

Applicable Features

The functions performed by the various security administrators through the trusted path are described in the System Administration Manual [SYAD]. Each administrator role is implemented as a special group that limits the functions available for that role to essential functions. An audit record is written when anyone assumes an administrator role.

Conclusion

Trusted XENIX satisfies the additional provisions of the B3 Trusted Facility Management requirement.[6]

3.21 Configuration Management

Requirement

During development and maintenance of the TCB, a configuration management system shall be in place that maintains control of changes to the descriptive top-level specification, other design data, implementation documentation, source code, the running version of the object code, and test fixtures and documentation. The configuration management system shall assure a consistent mapping among all documentation and code associated with the current version of the TCB. Tools shall be provided for generation of a new version of the TCB from source code. Also available shall be tools for comparing a newly generated version with the previous TCB version in order to ascertain that only the intended changes have been made in the code that will actually be used as the new version of the TCB.

[6]Although Trusted XENIX satisfies this requirement at the B3 level, it does not satisfy the assurance requirements above its rated level.

Applicable Features

When TIS obtained the Secure XENIX code from IBM, all code and documentation were placed under TIS's configuration control system. All changes to the descriptive top-level specification, all design documentation, implementation documentation, source code, object code, test fixtures, and other documentation are managed within the CM system under the control of the CCB.

Section 2.9.2 in the System Overview provides a detailed description of the configuration management process. Changing or adding Trusted XENIX code, hardware, or documentation requires submission of a Product Change Request (PCR) for CCB consideration. Before approving a PCR for implementation, the change is assessed relative to its impact on the rest of the system, and all impacts to code and documentation are identified. If the CCB gives its approval for implementing the change, a Change Tracking Report (CTR) is used to track the change through implementation and testing. If any further impacts are identified during implementation and testing, additional PCRs are submitted to accomplish these changes. Before a change is formalized and made a part of the CM Library, it must again be approved by the CCB.

The Source Code Control System (SCCS) provides automated tracking of all changes to code and documentation. SCCS maintains deltas between new and old versions and provides an automated tool for building any version of the TCB from source code. A newly generated version of the TCB can be compared with the previous version by using the *diff* command.

The standard Trusted XENIX protection mechanisms are used to prevent unauthorized changes to the SCCS files. The CM librarian has complete access to the SCCS database. General users can only read SCCS files. So while an ordinary user can read (copy) a file from SCCS, only the CM librarian can make the change to the information managed under SCCS. In addition to the Trusted XENIX protection mechanisms, each SCCS file maintains a checksum of its own contents. If a SCCS file becomes corrupted, the SCCS file will not be processed by the commands.

TIS has presented evidence of its configuration management process to the evaluation team. The procedures in place to support the CM process are described in section 2.9.2.

Conclusion

Trusted XENIX satisfies the B2 Configuration Management requirement.

3.22 Security Features User's Guide

Requirement

A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use, and how they interact with one another.

Applicable Features

The security features of Trusted XENIX are described in the *Trusted XENIX Security Features User's Guide* [SFUG]. This manual describes how to log onto the system and how to use the various protection mechanisms such as DAC and MAC. It also explains the use of labels and, in general, what an ordinary user needs to know to use the security features of Trusted XENIX. Establishing a secure communication link with the TCB is described. Also, the use of access control lists, which are used for finer access control than standard UNIX, is explained.

Conclusion

Trusted XENIX satisfies the B2 Security Features User's Guide requirement.

3.23 Trusted Facility Manual

Requirement

A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility. The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given. The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to securely generate a new TCB, and facility procedures, warnings, and privileges that need to be controlled in order to operate the facility in a secure manner. The TCB modules that contain the reference validation mechanism shall be identified. The procedures for secure generation of a new TCB from source after modification of any modules in the TCB shall be described.

Applicable Features

The Trusted Facility Manual for this system is composed of two documents: *Starting Trusted XENIX* [STXE] and *Trusted XENIX System Administration Manual* [SYAD]. These two documents describe the various operator, administrator, and privileged roles needed to operate a secure facility. They describe the separation of privileges available to the different roles and each role's interface functions, commands, and files. They provide guidelines for the use of these roles to install, run and maintain a Trusted XENIX installation. Some general cautions and warnings are also provided along with examples under each role description. These two documents include guidelines on the consistent and effective use of the protection features of the system, how they interact, and facility procedures, warnings, and privileges that need to be controlled in order to operate the facility in a secure manner. Specifically, they describe the physical security concern and describe alternatives. They also include the audit record structure, as well as details on how to examine and maintain the audit log.

The source code is not distributed so when any changes are made, a new TCB must be obtained from TIS. Because the source code is not distributed, neither the TCB modules which contain the reference validation mechanism nor the procedures for generating a new TCB from source are described in the either of the two manuals.

Conclusion

Trusted XENIX satisfies the B2 Trusted Facility Manual requirement.

3.24 Test Documentation

Requirement

The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechsnisms were tested, and results of the security mechanisms' functional testing. It shall include results of testing the effectiveness of the methods used to reduce covert channel bandwidths.

Applicable Features

TIS has provided a description of the method to be used to test the Trusted XENIX kernel and the detailed test plans for the kernel calls. The technique described (called the "Grey-Box Approach" [GLIG]) relies on analysis of dependencies to reduce the number of test cases that need to be exercised. TIS has analyzed Trusted XENIX version 2.0 and produced a kernel call control-synthesis graph, analyzed the dependencies, and produced the test plans for the kernel calls that its analysis showed to be necessary. The specific access checks of a kernel call has shown that the untested subpaths of that kernel call join the tested path.

For Trusted Process testing, TIS used the more traditional "Black-Box" approach. The test plans are comprehensive, clear, and usable.

TIS has analyzed and estimated covert channel bandwidths (see page 74 for more details).

Conclusion

Trusted XENIX satisfies the B2 Test Documentation requirement.

3.25 Design Documentation

Requirement

Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB. The interfaces between the TCB modules shall be described. A formal description of the security policy model enforced by the TCB shall be available and proven that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model. The descriptive top-level specification (DTLS) shall be shown to be an accurate description of the TCB interface. Documentation shall describe how the TCB implements the reference monitor concept and give an explanation why it is tamper resistant, cannot be bypassed, and is correctly implemented. Documentation shall describe how the TCB is structured to facilitate testing and to enforce least privilege. This documentation shall also present the results of the covert channel analysis and the tradeoffs involved in restricting the channels. All auditable events that may be used in the exploitation of known covert storage channels shall be identified. The bandwidths of known covert storage channels, the use of which is not detectable by the auditing mechanism, shall be provided.

Applicable Features

In Trusted XENIX the security policy model enforced by the TCB is the Bell and La Padula model. A description of TIS's philosophy of protection and an explanation of how this philosophy is translated into the TCB is provided in the Interpretation of the Bell and La Padula Model in Trusted XENIX [INTP]. This document also describes how the TCB protection mechanisms satisfy the model.

The TCB and the interfaces between the TCB modules are described in Trusted XENIX Architecture vols. 1 and 2 [KERN, TPAR]. A description of how the TCB enforces least privilege is provided in vol. 1. The interface to the TCB is described in the DTLS.

The descriptions in The Reference Monitor of Trusted XENIX [RFMN] accurately portray how the Trusted XENIX TCB correctly implements the reference monitor concept. A description of how the Trusted XENIX TCB is structured adequately to facilitate testing is provided in A New Security Testing Method and its Application to the Trusted XENIX Kernel [GLIG] and Trusted XENIX Test Plan vols. 1 and 2 [KTST, TTST].

The results of covert channel analysis is provided in Trusted XENIX Covert-Channel Identification: Potential Covert Channels [CCAID] and Trusted XENIX Covert Channel Capacity Estimation and Reduction [CCABW] and described elsewhere in this report (see section 3.19, page 95).

The kernel packaging is described in two appendices of the kernel architecture document [KERN].

Conclusion

Trusted XENIX satisfies the B2 Design Documentation requirement.

Chapter 4

Evaluator's Comments

- The methods Trusted XENIX uses to maintain ACLs can lead to confusion. The *acl* command, which is not accessible through the trusted shell, allows the user to manipulate ACLs while maintaining them in the sorted order that unambiguously defines their meaning (i.e., they are sorted in order from most specific to least specific). The *acl* system calls, however, which are part of the direct TCB interface do not maintain this sorted order. When the system calls are used, assuring that the desired protection is achieved by the proper ordering of the individual ACL components is the user's responsibility. Because the command and system call interfaces might both be used, any use with the system call should maintain the order of most-specific to least-specific.

- Trusted XENIX implements a very good trusted facility management mechanism. This mechanism supports five distinct roles–Trusted System Programmer, System Security Administrator, Auditor, Secure Operator, and Accounts Administrator, which perform role-related actions from within a restricted environment.

- Trusted XENIX does a poor job of making use of the available hardware mechanisms provided by the underlying system. This can be seen in multiple ways. Trusted XENIX utilizes only two of the four hierarchical rings supported by the Intel processors–one for the kernel and the other for trusted *and* untrusted subjects. Trusted XENIX implements only a single call gate to provide access to all kernel calls, rather than making use of multiple hardware-supported call gates to automatically control access to the kernel. Trusted XENIX could be structured to make more effective use of segmentation and other hardware protection features. This would enhance security and provide better internal structure, though at the expense of greater (internal) divergence from the original UNIX system on which it is based.

- Given the desktop (and portable, in the case of the GRiDCASE 1537) nature of some of the hardware bases of Trusted XENIX, it is likely that, in some cases, protecting the hardware portions of the TCB will be next to impossible or may simply be overlooked. While some machines provide fairly good boot protection and even cabinet locks, others provide no such protection whatsoever (see the TFM). In any case, it is important that this problem is understood, and even more critical that this problem is addressed for any installed product.

- The test plan for Trusted XENIX is very comprehensive and thorough. This is mainly credited to the *Grey Box* testing methodology.

- The system does not supply default ACLs for files or directories; instead, one must use the settings on the protection bits to initially create an ACL, and then manually add and delete entries to obtain the desired initial ACL.

This page intentionally left blank

Appendix A

Evaluated Hardware Components

Trusted XENIX is approved to run on a variety of hardware bases. This section is broken down into sections providing information regarding machines and components that utilize a MCA bus and machines and components that utilize an ISA bus. After each Central Unit section a set of components that are approved for use with that central unit(s) only are listed. A final section provides information regarding components that are bus and machine independent.

For each central unit utilizing a MCA bus it is assumed that the standard configuration is utilized, namely that the implementation includes the standard DMA Controller, Interrupt Controller, Timer, Real Time Clock, Keyboard Controller, Serial Port Controller, and Parallel Port Controller. Therefore the names of these individual components are not listed.

For each central unit utilizing an ISA bus the specific DMA Controller, Interrupt Controller, Timer, Real Time Clock, Keyboard Controller, Serial Port Controller, and Parallel Port Controller approved for use with the central unit is identified. Each of these central units are produced by a different vendor and utilize different devices thus there is not a standard configuration across the machines.

Due to performance, cost, or availability a vendor may choose to use a new device within their central units. However in order to be considered an evaluated configuration the central unit must contain either the standard configuration in the case of the MCA bus or the particular devices (described below) in the case of the ISA bus.

In addition each central unit must have associated with it some set of system integrity tests that can be run periodically to validate the correct operation of the on-site hardware. In most cases the vendors of the central unit provide a set of Power On Self Tests (POST) that provide a cursory examination of the hardware and a separate set of advanced diagnostics that more extensively test the hardware. In most cases the diagnostics provided by the vendor do not test the hardware to a sufficient degree and must be supplemented by TIS created diagnostics and by diagnostics from a third party. TIS includes a diagnostic package with Trusted XENIX that is designed to test the security machanisms of the base microprocessor. Version 3.0 of Check$\surd$It, by Touchstone Software Corporation 2130 Main Street, Suite 250, Huntington Beach, CA 92648, has been approved for use as a supplemental set of diagnostic tests for peripheral devices. Check$\surd$It can be obtained by contacting Touchstone directly at (800) 531-0450 or (714) 969-7746 or through various software retailers.

MCA Bus

Central Unit

- IBM PS/2 Model 80
- IBM PS/2 Model 70
- IBM PS/2 Model P70

- IBM PS/2 Model T70
- IBM PS/2 Model 60
- IBM PS/2 Model 50

The following components must be used with an MCA bus and are approved to be used interchangeably in the central units named above, namely the IBM PS/2 Model 50,60,70,70P,70T, and 80. Any exceptions to this rule are noted with footnotes.

Video Display Adapters

- IBM Display Adapter 8514/A
- IBM PS/2 Display Adapter

Fixed Disk Drive Adapters

- IBM Fixed Disk Drive Adapter/A, Types 1,2 (ST506)
- IBM ESDI Fixed Disk Drive Adapter/A (ESDI)
- IBM PS/2 60Mb Fixed Disk Drive Adapter (ESDI)[1]

Floppy Diskette Drive Adapters

- IBM PS/2 Diskette Drive Adapter

Additional communication Adapters

- Dual Async Adapter (PS/2)

Cartridge Tape Backup Unit Adapter

- Wangtek MCA Host Adapter, 33577-001

Memory Expansion Units

- IBM 80286 Memory Expansion
- IBM PS/2 2-8MB 80286 Memory Expansion

[1]The IBM PS/2 60Mb Fixed Disk Drive Adapter can be used only with the IBM PS/2 model 50.

- IBM 80386 Memory Expansion
- IBM 2-8MB 80386 Memory Expansion
- IBM PS/2 0-8MB Expanded Memory Adapter/A

ISA Bus

Central Unit

- IBM PC/AT 5170 Model 099
- IBM PC/AT 5170 Model 239
- IBM PC/AT 5170 Model 339

The following components are approved to be used interchangeably in the IBM PC AT Model 099, 239 and 339. Any exceptions to this rule are noted with footnotes.

Video Display Adapters

- IBM Monochrome Display and Printer Adapter
- IBM Color/Graphics Monitor Adapter
- TAXAN Monochrome Graphics Adapter with Parallel Port (MGP 256)

Fixed Disk Drive Adapters

- IBM Fixed Disk Adapter (ST506)
- IBM AT Fixed Disk and Diskette Drive Adapter (ST506)
- Adaptec ACB2310/12 (ST506)
- Western Digital WD1003-WA2 (ST506)

Floppy Diskette Drive Adapters

- IBM AT Diskette Drive Adapter
- IBM AT Fixed Disk and Diskette Drive Adapter

DMA Controller

- Intel 8237 DMA Controller

Interrupt Controller

- Intel 8259 Interrupt Controller

Timer

- Intel 8254 Timer

Real Time Clock

- Motorola MC146818AP

Keyboard Controller

- Intel 8742 UPI

Serial Port Controller

- IBM Serial and Parallel Port Adapter

Parallel Port Controller

- IBM Serial and Parallel Port Adapter

Central Unit

- AST 386/25

The following components are approved to be used in the AST 386/25 Mhz.

Video Display Adapters

- Paradise Western Digital PVGA1A Controller

Fixed Disk Drive Adapters

- Conner CP 3104 IDE Drive (ST506)[2]

[2]Conner Peripheral Drives provide an integrated hard disk drive and hard disk drive controller.

Floppy Diskette Drive Adapters

- Western Digital WD37C65/A/B Floppy Disk Subsystem Controller

DMA Controller

- Chips & Technology 82C206 Integrated Peripheral Controller

Interrupt Controller

- Chips & Technology 82C206 Integrated Peripheral Controller

Timer

- Chips & Technology 82C206 Integrated Peripheral Controller

Real Time Clock

- Chips & Technology 82C206 Integrated Peripheral Controller

Keyboard Controller

- AST Keyboard Controller 237002-001

Serial Port Controller

- National Semiconductor 16450

Parallel Port Controller

- AST ACORN Parallel Port Subsystem

Central Unit

- Grid 1537

The following components are approved to be used in the Grid 1537.

Video Display Adapters

- Chips & Technology F82C455 Flat Panel/CRT VGA Controller

Fixed Disk Drive Adapters

- Conner CP344 (ST506)[3]

Floppy Diskette Drive Adapters

- National Semiconductor DP 8473 Floppy Disk Controller PLUS-2

DMA Controller

- Chips & Technology 82C206 Integrated Peripheral Controller

Interrupt Controller

- Chips & Technology 82C206 Integrated Peripheral Controller

Timer

- Chips & Technology 82C206 Integrated Peripheral Controller

Real Time Clock

- Chips & Technology 82C206 Integrated Peripheral Controller

Keyboard Controller

- Intel 8742 Keyboard Controller

Serial Port Controller

- VLSI Technology VL16452

Parallel Port Controller

- VLSI Technology VL16452

[3]Conner Peripheral Drives provide an integrated hard disk drive and hard disk drive controller.

Central Unit

- NECP 386/25 PowerMate
- NECB 386/25 BusinessMate

The following components are approved to be used in the NECP 386/25 PowerMate and NECB 386/25 BusinessMate.

Video Display Adapters

- Paradise Western Digital PVGA1A Controller

Fixed Disk Drive Adapters

- Western Digital 1007A WAH[4]

Floppy Diskette Drive Adapters

- NEC D72068F6F

DMA Controller

- Chips & Technology 82C206 Integrated Peripheral Controller

Interrupt Controller

- Chips & Technology 82C206 Integrated Peripheral Controller

Timer

- Chips & Technology 82C206 Integrated Peripheral Controller

Real Time Clock

- Chips & Technology 82C206 Integrated Peripheral Controller

Keyboard Controller

[4]The Western Digital 1007A WAH Controller provides an ST506 interface to the CPU but provides an ESDI interface with the hard disk drive, thus it must be used with an ESDI hard disk drive.

- NEC APC IV Series Keyboard Controller

Serial Port Controller

- VLSI Technology VLSI 16C452

Parallel Port Controller

- VLSI Technology VLSI 16C452

Central Unit

- Unisys Personal Workstation 2 Series 800/20C

The following components are approved to be used in the Unisys Personal Workstation 2.

Video Display Adapters

- Headland GC205-PC VGA Controller

Fixed Disk Drive Adapters

- NCR SCSI Protocol Controller 5380

Floppy Diskette Drive Adapters

- NCL 20-20-303 (This is the chip found on the NCR SCSI Controller 5380)

DMA Controller

- Zymos 82C30 Poach Integrated Peripheral Controller

Interrupt Controller

- Zymos 82C30 Poach Integrated Peripheral Controller

Timer

- Zymos 82C30 Poach Integrated Peripheral Controller

Real Time Clock

- Zymos 82C30 Poach Integrated Peripheral Controller

Keyboard Controller

- Phoenix Keyboard Controller

Serial Port Controller

- Western Digital 16C452

Parallel Port Controller

- Western Digital 16C452

Central Unit

- Zenith Z-386/33

The following components are approved to be used in the Zenith Z-386/33.

Video Display Adapters

- Headland GC208-PC VGA Controller

Fixed Disk Drive Adapters

- Data Tech Corp DTC 455C 05302[5]

Floppy Diskette Drive Adapters

- Data Tech Corp DTC 455C 05302

DMA Controller

- Chips & Technology 82C206 Integrated Peripheral Controller

[5]The DTC 455C 05302 Controller provides an ST506 interface to the CPU but provides an ESDI interface with the hard disk drive, thus it must be used with an ESDI hard disk drive.

Interrupt Controller

- Chips & Technology 82C206 Integrated Peripheral Controller

Timer

- Chips & Technology 82C206 Integrated Peripheral Controller

Real Time Clock

- Chips & Technology 82C206 Integrated Peripheral Controller

Keyboard Controller

- Zenith Z386 Keyboard Controller

Serial Port Controller

- VLSI Technology 16C452

Parallel Port Controller

- VLSI Technology 16C452

The following components designed to be used with an ISA bus and are approved to be used with any of the central units list above that provide an ISA bus. Any exceptions to this rule are noted with footnotes.

Memory Expansion Units, ISA Bus and Machine Independent

- IBM 128KB/640KB Memory Expansion
- IBM 512KB/2MB Memory Adapter
- IBM 256KB Memory Expansion

Cartridge Tape Backup Unit Adapter

- Wangtek PC-02 Host Adapter, 30631-001

The following components are Bus Independent and Machine Independent and are approved to be used with any of the central units listed above. Any exceptions to this rule are noted with footnotes.

Fixed Disks

- IBM PC/AT 30MB Fixed Disk (ST506)
- IBM 3.5 inch 20MB Fixed Disk (ST506)
- IBM 5.25 inch 44MB Fixed Disk (ST506)
- CDC Swift (94355-100) (ST506)
- IBM 5.25 inch 70MB Fixed Disk (ESDI)
- IBM 5.25 inch 115MB Fixed Disk (ESDI)
- IBM 5.25 inch 314MB Fixed Disk (ESDI)
- IBM 3.5 inch 60MB Fixed Disk (ESDI)
- IBM 3.5 inch 120MB Fixed Disk (ESDI)
- IBM 3.5 inch 30MB Fixed Disk (ESDI)
- Miniscribe 3180E 155MB (ESDI)
- NEC P/N 134-500535-115 339 MB (ESDI)
- Quantum Pro Drive Series 40 MB (SCSI)

Floppy Diskette Drives

Basically any Floppy Diskette Drive that supports the IBM AT Standard Interface can be used with the system. Some examples of drives that have been used by TIS include the following:

- IBM 1.2 MB 5.25" Floppy Disk Drive
- IBM 1.44 MB 3.5" Disk Drive
- IBM 360 KB 5.25" Floppy Disk Drive
- IBM 720 KB 3.5" Disk Drive
- Toshiba 1.2 MB 5.25" Fl. Drv. ND-08DE-6
- Mitsubishi 1.2 MB 5.25" Fl. Drv. MF504B-318

- TEAC 1.44 MB 3.5" Fl. Drv. FD-235HF
- TEAC 1.44 MB 3.5" Fl. Drv. FD-216HF
- NEC 1.2 MB 5.25" Fl. Drv. P/N 134-500357-007-0
- SONY 1.44 MB 3.5" Fl. Drv. MT-F17W37D
- SONY 1.44 MB 3.5" Fl. Drv. MT-F17W42D

Monitors

- CGA Color/Monochrome Monitors
- EGA Color/Monochrome Monitors
- Monochrome Monitors
- MCGA Color/Monochrome Monitors
- VGA Color/Monochrome Monitors

Math Co-Processors

- INTEL 80287 Math Co-processor (use with 286 based machines)
- INTEL 80387 Math Co-processor (use with 386 based machines)

Printers[6]

- NEC Spinwriter 8800
- Panasonic KX1191
- Star NX-1000
- IBM Proprinter II

Cartridge Tape Backup Unit

- Wangtek Cartridge Tape Unit 5125K/5150PK

[6] Trusted XENIX filters all data that goes to the printer such that only the hex characters 00 (null), 08 (backspace), 09 (horizontal tab), 0A (new line), 0C (form feed), 0D (carriage return), and 20 through 7E (standard ASCII characters) are sent to the printer. Hence, any printer that simply prints (as oppossed to interpreting as some command) this set of characters is acceptable in the evaluated configuration).

In addition, the following components may be used with the system:

- Any ASCII terminals
- Any RS232 cables
- Any printer cables

This page intentionally left blank

Appendix B

Evaluated Software Components

Trusted XENIX version 3.0

This page intentionally left blank

Appendix C

Acronyms

The following table contains a list of acronyms that are used in the report.

Acronym	Definition
AA	Accounts Administrator
ACL	Access Control List
ADP	Automated Data Processing
BIOS	Basic Input/Output System
CCB	Change Configuration Board
CI	Configuration Item
CM	Configuration Management
CMOS	Complementary Metal-Oxide Semiconductor
CPL	Current Privilege Level
CPL	Current Process Level
CPU	Central Processing Unit
CS	Code Segment Selector
CTR	Change Tracking Report
DAC	Discretionary Access Control
DMA	Direct Memory Access
DPL	Descriptor Privilege Level
DS	Data Segment Selector
DTLS	Descriptive Top Level Specification
DoD	Department of Defense
EOF	End of File
EPL	Evaluated Products List
ES	Extra Segment Selector
GDT	Global Descriptor Table
GDTR	Global Descriptor Table Register
GPM	Generalized Privilege Mechanism
IDT	Interrupt Descriptor Table
IOPL	Input/Output Privilege Level
IP	Instruction Pointer
IPC	Inter-process communication
KB	Kilobyte
LDT	Local Descriptor Table
MAC	Mandatory Access Control
MB	Megabyte

Acronym	**Definition**
MHz	Megahertz
NCSC	National Computer Security Center
NMI	Non-Maskable Interrupt
OS	Operating System
PCR	Product Change Request
PC	Personal Computer
ROM	Read Only Memory
SAK	Secure Attention Key
SCCS	Source Code Control System
SFUG	Security Features Users' Guide
SO	Secure Operator
SS	Stack Segment Selector
SSA	System Security Administrator
TCB	Trusted Computing Base
TCSEC	Trusted Computer System Evaluation Criteria
TFM	Trusted Facility Manual
TP	Trusted Process
TSH	Trusted Shell
TSP	Trusted Systems Programmer
TSS	Task State Segment

This page intentionally left blank

Appendix D

Bibliography and References

[286 85] *iAPX 286 Programmer's Reference Manual* Intel Corporation, Order Number 210498, 1985.

[386 86] *80386 Programmer's Reference Manual* Intel Corporation, Order Number 230985-001, 1986.

[ASTT] "AST Premium Technical Reference", AST Research Inc., October 1990.

[ASTT] "AST Premium User's Manual", AST Research Inc., October 1990.

[BAUE] *Security Concepts for Microprocessor Based Key Generator Controllers*, SYTEK TR-84009, Bauer, R. K., R. J. Feiertag, B. L. Kahn and W. F. Wilson, April 24, 1984.

[BELL] Bell, David E. and LaPadula, Leonard J., *Secure Computer Systems: Unified Exposition and Multics Interpretations*, MTR-2997, rev 1, The MITRE Corp., Bedford, MA, March 1976.

[CCAID] *Trusted XENIX Covert-Channel Identification: Potential Covert Channels* Trusted Information Systems, Inc., Glenwood, MD, 1992 (Company Proprietary).

[CCABW] *Trusted XENIX Covert Channel Capacity Estimation and Reduction* Trusted Information Systems, Inc., Glenwood, MD, 1992 (Company Proprietary).

[CHECKIT] "Check√It PC Diagnostic Software User Guide", Touchstone Software Corporation, October 1990.

[CMREF 1] *TIS Trusted XENIX Commands Reference Volume 1*, Trusted Information Systems, Inc., version 3.0, 1992.

[CMREF 2] *TIS Trusted XENIX Commands Reference Volume 2*, Trusted Information Systems, Inc., version 3.0, 1992.

[DIJK] Dijkstra, E. W., *Cooperating Sequential Processes* in Programming Languages, ed. F. Genuys, Academic Press, New York, NY, 1968.

[DTLS] *Trusted XENIX DTLS*, Trusted Information Systems, Inc., version 3.0, 1992.

[GLIG] Gligor, V. D., Chandersekaran, C. S., et al., *A New Security Testing Method and its Application to the Secure XENIX Kernel* Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, CA, April, 1986.

[GRIDU] "GRiDCASE 1537 User's Guide", GRiD Systems Corporation, May 1990.

[GRIDT] "GRiDCASE 1537 Technical Reference Manual", GRiD Systems Corporation, June 1991.

[INST] "IBM PC/AT Installation and Setup".

[INTP] *Interpretation of the Bell and LaPadula Model in Trusted XENIX*, Trusted Information Systems, Inc..

[KERN] *Trusted XENIX Architecture Vol 1: Kernel*, Trusted Information Systems, Inc., version 2.0, March 9, 1990.

[KTST] *Trusted XENIX Test Plan: Volume I, Kernel Calls*, Trusted Information Systems, Inc., version 1.1, May 16, 1987.

[MAIN] *Personal Computer Hardware Reference Library Hardware Maintenance and Service (two volumes)*, Trusted Information Systems, Inc., Manual Number 1502493, March 1984.

[NECB] "BusinessMate 386/25 System Installation Guide", NEC Technologies Inc., November 1989.

[NECP] "PowerMate 386/25 Owner's Guide", NEC Technologies Inc., October 1989.

[RFMN] *The Reference Monitor of Secure XENIX*, IBM, ver 1.0, October 14, 1986.

[SFUG] *Trusted XENIX Security Features User's Guide*, Trusted Information Systems, Inc., version 3.0, 1992.

[SIBE] Sibert, W. O., *et al*, *UNIX and B2 — Are They Compatible?* Proceedings of the 10th National Computer Security Conference, 21–24 September 1987.

[STXE] *Starting Trusted XENIX*, Trusted Information Systems, Inc., version 3.0, 1992.

[SVID] *System V Interface Definition* Spring 1985, Issue 1, AT&T Customer Information Center, Indianapolis, IN.

[SYAD] *Trusted XENIX System Administration Manual* Trusted Information Systems, Inc., version 3.0, 1992.

[SYSREF] *TIS Trusted XENIX System Reference*, Trusted Information Systems, Inc., version 3.0, 1992.

[TFM] *Trusted XENIX Trusted Facility Manual*, Trusted Information Systems, Inc., version 3.0, 1992.

[TPAR] *Trusted XENIX Architecture Vol 2: Trusted Processes*, Trusted Information Systems, Inc., version 1.8, January 30, 1990.

[TSAI] Tsai, C. R., Gligor, V. D., Chandersekaran, C. S., *A Formal Method for the Identification of Covert Storage Channels in Secure XENIX*, Proceedings of the 1987 IEEE Symposium on Security and Privacy, Oakland, CA, April, 1987.

[TTST] *Trusted XENIX Test Plan: Volume II, Trusted Processes*, Trusted Information Systems, Inc., version 1.3, June 22, 1988.

[TECH] *Personal Computer Hardware Reference Library Technical Reference*, Trusted Information Systems, Inc., Manual Number 1502494, March 1984.

[UNIPWI] "Personal Workstation *2* Series 800/20C Installation Guide", Unisys Corporation, January 1990.

[UNIPWT] "Personal Workstation *2* Series 800/20C Engineering Reference Manual", Unisys Corporation, January 1990.

[ZENTI] "Z-386/AT Series Workstation Technical Reference Manual Volume I Hardware and BIOS", Zenith Data Systems Corporation, 1989.

[ZENTII] "Z-386/AT Series Workstation Technical Reference Manual Volume II Programmable Registers, Zenith Data Systems Corporation, 1989.

[ZENTIII] "Z-386/AT Series Workstation Technical Reference Manual Volume III Programmable Registers BIOS", Zenith Data Systems Corporation, 1989.